Class Notes in Computer Science (First Edition)

Goodwin, Roger L

25 September 2015

Class Notes in Computer Science (First Edition, v1)

Edited by Roger L. Goodwin

Summit Point, WV 25446

Editor's Note: In life, it became too cumbersome for me to look-up equations, theorems, proofs, and problem solutions from prior courses. I had over three boxes of notes. That is when I decided to typeset them. I have been doing this for over two decades. These notes can be downloaded for *free* from the web site http://www.repec.org/. Note that the beginning of each chapter lists the professor's name and affiliation. Additionally, the course number, the date the course was taken, and the text book are given.

Contents

1 Discrete Structures (L. J. Randall) 1

1.1 Mathematical Models and Reasoning . . . 2
1.1.1 Hand-Out of Reasoning Problems . . . 2
1.1.2 Introduction . . . 3
1.1.3 Mathematical Reasoning . . . 3
1.1.4 Homework and Answers . . . 5
1.1.5 Predicates and Quantifiers . . . 7
1.1.6 Quiz 1 . . . 9
1.1.7 Quantifiers with Negation . . . 10
1.1.8 Homework and Answers . . . 12
1.1.9 Homework and Answers . . . 13
1.1.10 Quiz 2 . . . 14
1.1.11 Logical Inferences . . . 15
1.1.12 Handout . . . 17
1.1.13 Homework Handout Questions and Answers . . . 18
1.1.14 Methods of Proof . . . 19
1.1.15 Proof Techniques for Quantified Assertions . . . 21
1.1.16 Homework and Answers . . . 22
1.2 Sets . . . 23
1.2.1 Exam 2 . . . 24
1.2.2 Quiz 3 . . . 25
1.2.3 Operations on Sets . . . 26
1.2.4 Homework and Answers . . . 27
1.2.5 Homework and Answers . . . 27
1.2.6 Homework and Answers . . . 28
1.2.7 Complement of Sets . . . 29
1.2.8 Venn Diagrams . . . 29
1.2.9 Homework and Answers . . . 32
1.2.10 Generalized Unions and Intersections . . . 35
1.2.11 Induction . . . 35
1.2.12 Inductive Definition of Sets . . . 36
1.2.13 Set Operations on Σ^* . . . 38
1.2.14 Homework and Answers . . . 38
1.3 Binary Relations . . . 41
1.3.1 Graph Theory . . . 41
1.3.2 Homework and Answers . . . 44
1.3.3 Homework and Answers . . . 45
1.3.4 Composition of Relations . . . 46
1.3.5 Closure Operations on Relations . . . 46
1.3.6 Homework and Answers . . . 48
1.3.7 Homework and Answers . . . 48
1.3.8 Homework and Answers . . . 48

1.3.9 Quiz 4 . . . 50
1.3.10 Exam 3 . . . 51
1.3.11 Quiz 5 . . . 53

2 Theory of Formal Languages (L. J. Randall) 55
2.1 Overview of the Course . . . 55
2.2 Turing Machines . . . 57
2.3 The Foundational Programming Languages . . . 58
2.3.1 The Goto Language . . . 59
2.3.2 Homework and Answers . . . 60
2.3.3 The While Language . . . 63
2.3.4 Flowcharts . . . 63
2.3.5 Homework and Answers . . . 64
2.3.6 Homework and Answers . . . 65
2.3.7 Exam 1 and Answers . . . 70
2.4 Functions . . . 71
2.4.1 Computable Functions . . . 72
2.4.2 Homework and Answers . . . 75
2.4.3 Primitive Recursion . . . 76
2.4.4 Mu Recursion . . . 77
2.4.5 Homework and Answers . . . 79
2.4.6 Formal Computations Summary . . . 80
2.5 Context Free Grammars . . . 81
2.5.1 Exam 2 and Answers . . . 84
2.5.2 Lambda Productions . . . 85
2.5.3 Homework and Answers . . . 87
2.5.4 Homework and Answers . . . 87
2.5.5 Parsing . . . 88
2.6 Regular Languages and Finite Automata . . . 90
2.6.1 Regular Expressions . . . 93
2.6.2 Pushdown Automata . . . 94
2.7 Homework and Answers . . . 97
2.8 Homework and Answers . . . 99
2.9 Homework and Answers . . . 99
2.10 Handout and Answers . . . 99

A Index 103

Chapter 1

Discrete Structures

Place: Old Dominion University
Timeframe: Fall Semester, 1987
Instructor: Jane Randall
Office: ED 256-7
Phone: 440-3890 (my office)
Phone: 440-3915 (CS office)

Office Hours: Monday: 3:00 - 5:00, Wednesday: 11:00 - 12:00, Friday: 10:00 - 12:00

Course Description: This course will cover a variety of discrete mathematical concepts which have applications in computer science. Topics will include symbolic logic, set theory, binary relations, and functions.

Prerequisite: CS 160

Text: *Discrete Mathematics in Computer Science,* by Stanat and McAllister.
Material to be covered: Chapters 0 through 3, and if time permits, part of chapter 4.

Grading: Your grade will be based on the following –

1. Four tests (including the final) 80% (20% each)
2. Several quizes 20%

Make-up tests: A test may be made up only if I am contacted within 24 hours after the test is given and if the reason for absence is legitimate.

Attendance plolicy: Students are expected to attend class. A student who must miss class is expected to obtain the assignment and be prepared for the next class meeting.

Honor code: All students are expected to abide by the ODU Honor Code. This means that all exams submitted are to be the exclusive work of the student. An honor pledge will be required on all work which is graded.

1.1 Mathematical Models and Reasoning

1.1.1 Hand-Out of Reasoning Problems

1. (a) All Students who major in philosophy wear Calvert Kreem jeans.

   (b) None of the students in the Marching and Chowder Society wears Clavert Kreem jeans or majors in history.

   (c) If Jack majors in philosophy, Mary majors in history.

   Question: If the statements above are all true, which of the following must also be true?

   (A) If Jack majors in philosophy, Mary does not wear Calvert Kreem jeans.

   (B) None of the students in the Marching and Chowder Society majors in philosophy.

   (C) If Jack wears Calvert Kreem jeans, he majors in philosophy.

   (D) If Mary majors in history, Jack is not in the Marching and Chowder Society.

   (E) Either Jack or Mary wears Calvert Kreem jeans.

   Question: The conclusion "Jack does not major in philosophy" could be validly drawn from the statements above if it were established that

   I. Mary does not major in history.

   II. Jack does not belong to the Marching and Chowder Society.

   III. Jack does not wear Calvert Kreem jeans.

   (A) I only, (B) II only, (C) III only, (D) I and III, (E) II and III

2. All good athletes want to win, and all athletes who want to win eat a well-balanced diet; therefore, all athletes who do not eat a well-balanced diet are bad athletes.

   Question: If the argument above is valid, then which of the following statements must be true?

   (A) No bad athlete wants to win.

   (B) No athlete who does not eat a well-balanced diet is a good athlete.

   (C) Every athlete who eats a well-balanced diet is a good athlete.

   (D) All athletes who want to win are good athletes.

   (E) Some good athletes do not eat a well-balanced diet.

   Question: Which of the following, if true, would weaken the argument above?

   (A) Ann wants to win, but she is not a good athlete.

   (B) Bob, the accountant, eats a well-balanced diet, but he is not a good athlete.

   (C) All the players on the Burros baseball team eat a well-balanced diet.

   (D) No athlete who does not eat a well-balanced diet wants to win.

   (E) Cindy, the basketball star, does not eat a well-balanced diet, but she is a good athlete.

1.1.2 Introduction

Mathematical modeling is a model which is an analogy for an object, phenomenon, or process. A mathematical model is a model based on mathematically stated principles. Because of the rigor and lack of ambiguity, it provides a good means for expressing *principles.* It has three components:

1. The phenomena or process that is to be modeled.
2. A math structure which is used to make assertions about the object being modeled.
3. A correspondence between the real world object and the math structure.

Example: Consider the position function $f(t) = 4t^2 + t + 16$ where t represents time and $f(t)$ represents position.

Here is a list of reasons for using models.

1. To present information in a form which is easily understood.
2. To provide a means for simplifying computations.
3. To predict parameter values for events which have not yet occurred.

The value of a model is best measured by its ability to answer questions and to make predictions about the object being modeled.

1.1.3 Mathematical Reasoning

A *proposition* is a statement which is either true or false but not both. We say that the true value of the statement is True or False.

Example: $2 + 4 \leq 3$ is False.

Example: 0 is an integer is True.

Example: $2x = 8$ is not a proposition.

Example: "Close the door" is not a proposition.

Example: "This sentence is false" is not a proposition because it is both True and False.

Notation: We use capital letters beginning with P to denote arbitrary propositions. The following is a list of operations on propositions.

1. Negation or not is denoted by $\neg$. The *truth table* is:

P	$\neg P$
T	F
F	T

   The value True can also be defined as True $\equiv 1$ and the value False can be defined as False $\equiv 0$.

2. Conjunction (i.e. and) is denoted by $\wedge$. The truth table is:

P	Q	$P \wedge Q$
0	0	0
0	1	0
1	0	0
1	1	1

3. Disjunction (i.e. or) is denoted by $\vee$. The truth table is:

P	Q	$P \vee Q$
0	0	0
0	1	1
1	0	1
1	1	1

4. Exclusive Or is denoted by $\oplus$. The truth table is:

P	Q	$P \oplus Q$
0	0	0
0	1	1
1	0	1
1	1	0

5. Implication is denoted by $\Rightarrow$. The truth table is:

P	Q	$P \Rightarrow Q$
0	0	1
0	1	1
1	0	0
1	1	1

   $Q \Rightarrow P$ is called the *converse* of $P \Rightarrow Q$.

   $\neg Q \Rightarrow \neg P$ is called the *contra-positive* of $P \Rightarrow Q$.

   $\neg P \Rightarrow \neg Q$ is called the *inverse* of $P \Rightarrow Q$.

6. Equivalence is denoted by $\Leftrightarrow$. The truth table is:

P	Q	$P \Leftrightarrow Q$
0	0	1
0	1	0
1	0	0
1	1	1

   Note that $P \Leftrightarrow Q$ means the same thing as $(P \Rightarrow Q) \wedge (Q \Rightarrow P)$.

P	Q	$P \Rightarrow Q$	$Q \Rightarrow P$	$(P \Rightarrow Q) \wedge (Q \Rightarrow P)$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	1	1	1

Here are some more properties. $(P \vee \neg Q) \Rightarrow \neg P$.

P	Q	$\neg Q$	$P \vee \neg Q$	$\neg P$	$(P \vee \neg Q) \Rightarrow \neg P$
0	0	1	1	1	1
0	1	0	0	1	1
1	0	1	1	0	0
1	1	0	1	0	0

The *distributive property* is $[P \wedge (Q \vee R)] \Leftrightarrow [(P \wedge Q) \vee (P \wedge R)]$.

P	Q	R	$Q \vee R$	$P \wedge (Q \vee R)$	$P \wedge Q$	$P \wedge R$	$(P \wedge Q) \vee (P \wedge R)$
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

A *tautology* is a proposition which is always true. A *contridiction* is a proposition which is always false. A *contigency* is a proposition which is not a tautology and is not a contradiction. It is true part of the time and false part of the time.

Identities on page 15 of the text book. 7, 8, 18, 22. Delete 20 and 21. Logical implications on page 16 of the text book. Delete 7, 8, and 9. For homework, on page 17 in Section 1.1 of the text book, do problems 1, 3, 4, 5, and 7. Also show $(P \Leftrightarrow Q) \Leftrightarrow [(P \Rightarrow Q) \wedge (Q \Rightarrow P)]$.

1.1.4 Homework and Answers

These are the homework problems from Section 1.1 on pages 17 and 18 in the textbook.

Show $(P \Leftrightarrow Q) \Leftrightarrow [(P \Rightarrow Q) \wedge (Q \Rightarrow P)]$.

P	Q	$P \Leftrightarrow Q$	$P \Rightarrow Q$	$Q \Rightarrow P$	$(P \Rightarrow Q) \wedge (Q \Rightarrow P)$
0	0	1	1	1	1
0	1	0	1	0	0
1	0	0	0	1	0
1	1	1	1	1	1

3. Establish whether the following propositions are tautologies, contigencies, or contradictions.

a) $P \vee \neg P$ Tautology.
b) $P \wedge \neg P$ Contridiction.
c) $P \Rightarrow \neg(\neg P)$

P	$P \Rightarrow \neg(\neg P)$	
0	0	1
1	1	0

$\Rightarrow$ contingency

d) $\neg(P \wedge Q) \Leftrightarrow (\neg P \vee \neg Q)$. Tautology.

e) $\neg(P \vee Q) \Leftrightarrow (\neg P \wedge \neg Q)$. Tautology.

f) $(P \Rightarrow Q) \Leftrightarrow (\neg Q \Rightarrow \neg P)$. Tautology.

g) $(P \Rightarrow Q) \wedge (Q \Rightarrow P)$.

$(P \Rightarrow Q) \wedge (Q \Rightarrow P)$						
0	1	0	1	0	1	0
0	1	1	0	1	0	0
1	0	0	0	0	1	1
1	1	1	1	1	1	1

$\Rightarrow$ contingency

h) $[P \wedge (Q \vee R)] \Rightarrow [(P \wedge Q) \vee (P \wedge R)]$. Tautology.

i) $(P \wedge \neg P) \Rightarrow Q$. Taugology.

j) $(P \vee \neg Q) \Rightarrow Q$.

P	Q	$P \vee \neg Q$	$(P \vee \neg Q) \Rightarrow Q$	
0	0	1	0	
0	1	0	1	$\Rightarrow$ contingency
1	0	1	0	
1	1	1	1	

k) $P \Rightarrow (P \vee Q)$.

P	Q	$P \vee Q$	$(P \Rightarrow (P \vee Q)$	
0	0	0	1	
0	1	1	1	$\Rightarrow$ tautology
1	0	1	1	
1	1	1	1	

l) $(P \wedge Q) \Rightarrow P$.

P	Q	$P \wedge Q$	$(P \wedge Q) \Rightarrow P$	
0	0	0	1	
0	1	0	1	$\Rightarrow$ tautology
1	0	0	1	
1	1	1	1	

m) $(P \wedge Q) \Leftrightarrow P] \Leftrightarrow [P \Leftrightarrow Q]$

P	Q	$P \wedge Q$	$(P \wedge Q) \Leftrightarrow P$	$P \Leftrightarrow Q$	$[(P \wedge Q) \Leftrightarrow P] \Leftrightarrow [P \Leftrightarrow Q]$	
0	0	0	1	1	1	
0	1	0	1	0	0	$\Rightarrow$ contingency
1	0	0	0	0	1	
1	1	1	1	1	1	

n) $[(P \Rightarrow Q) \vee (R \Rightarrow S)] \Rightarrow [(P \vee R) \Rightarrow (Q \vee S)]$.

$$[(\neg P \vee Q) \vee (\neg R \vee S)] \Rightarrow [\neg(P \vee R) \vee (Q \vee S)],$$

$$\neg[(\neg P \vee Q) \vee (\neg R \vee S)] \vee [\neg(P \vee R) \vee (Q \vee S)],$$

$$[\neg(\neg P \vee Q) \wedge \neg(\neg R \vee S)] \vee [(\neg P \wedge \neg R) \vee (Q \vee S)],$$

$$[(P \wedge \neg Q) \wedge (R \wedge \neg S)] \vee [(\neg P \wedge \neg R) \vee (Q \vee S)].$$

Complete the truth table. Contingency.

4. Let P be the proposition "It is snowing." Let Q be the proposition "I will go to town." Let R be the proposition "I have time." Using logical connectives, write a proposition which symbolizes each of the following:

4a. i) If it is not snowing and I have time, then I will go to town. $(\neg P \wedge R) \Rightarrow Q$,

ii) I will go to town only if I have time. $Q \Rightarrow R$.

iii) It isn't snowing. $\neg P$.

iv) It is snowing and I will not go to town. $P \wedge \neg Q$.

4b. i) $Q \Leftrightarrow (R \wedge \neg P)$. I will go to town is equivalent to I have time and it is not snowing.

ii) $R \wedge Q$. I have time, and I will go to town.

iii) $(Q \Rightarrow R) \wedge (R \Rightarrow Q)$. If I will go to town then I have time, and If I have time then I will go to town.

iv) $\neg(R \vee Q)$. I don't have time and I won't go to town.

5. State the converse and contrapositive of each of the following:

a. If it rains, I'm not going. Solution: If I'm not going, it rains. If you don't go then it won't rain.

b. I will stay only if you go. Solution: If you go then I will stay. If you don't go then I won't stay.

c. I you get 4 pounds, you can bake the cake. Solution: If you bake the cake, you get 4 pounds. If you don't bake the cake then you won't get 4 pounds.

d. I can't complete the task if you don't get more help. Solution: If I can't complete the task, then I don't get more help. If I can complete the task, then I get more help.

7. Establish the following tautologies by simplifying the left side to the form of the right side:

a. $[(P \wedge Q) \Rightarrow P] \Leftrightarrow 1$ Solution:
$\neg(P \wedge Q) \vee P$
$(\neg P \vee \neg Q) \vee P$
$(\neg P \vee P) \vee \neg Q$
$1 \vee \neg Q$
1

b. $\neg(\neg(P \vee Q) \Rightarrow \neg P) \Leftrightarrow 0$, Solution:
$\neg((P \vee Q) \vee \neg P)$,
$\neg(P \vee Q) \wedge P$,
$(\neg P \wedge \neg Q) \wedge P$,
$(\neg P \wedge P) \wedge \neg Q$,
$0 \wedge \neg Q$,
0.

1.1.5 Predicates and Quantifiers

Quiz on Section 1.1 on Monday. It will cover truth tables, tautologies, etc. Don't have to know identities.

The statement $P \Leftrightarrow Q$ can be re-stated several ways, such as P if and only if Q or P iff Q.

A *predicate* is a property of an object or a relationship between objects.

Example: $x < y$ means x is less than y where the phrase "is less than y" is the predicate. "x is an integer is an integer" where the phrase "is an integer" is a predicate.

Notation: Assertions made with predicates are denoted with capital letters.

Example: x is less than $y \rightarrow L(x, y)$.

Example: x is tall $\rightarrow T(x)$.

Example: $x + y = z \rightarrow S(x, y, z)$.

The predicate (P) $P(x_1, x_2, ..., x_n)$ is said to have n arguments with individual variables $x_1, x_2, ..., x_n$. Values for the variables are drawn from a non-empty set called the *universe,* denoted by U.

Example: Let $P(x, y)$ mean $x \leq y$ where U is the set of integers. Then, $P(3, 5)$ is True. $P(6, 6)$ is True. $P(6, 2)$ is False.

Example: Let $S(x, y)$ mean $x - y = 5$ and U be the set of integers. Then, $S(12, 7)$ is True. $S(4, -2)$ is False.

To change a predicate into a proposition, each individual variable must be *bound.* There are two ways to do this:

1. A variable is bound when a value is assigned to it.
2. A variable is bound when it is *quantified.*

This is a list of quantifiers:

1. The universal quantifier $\forall$ (for all). $\forall x\ P(x)$ means for all x, $P(x)$ is true if the predicate $P(x)$ is true for all x in the universe. Otherwise, $\forall x\ P(x)$ is false.

   Example: Let U be the set of integers. $\forall x\ [x > 0]$ is False.

   Example: Let U be the set of natural numbers $\{1, 2, 3...\}$. $\forall x\ [x > 0]$ is True.

   Example: Let U be the set of integers. $\forall x \forall y\ [x + y > x]$ is False.

   Example: Let U be the set of integers. $\forall x \forall y\ [2(x + y) = 2x + 2y]$ is True.

   Note that $\forall x\ P(x) \Rightarrow P(c)$ is true for any c in the universal set. The notation $\{c \in U\}$ is also used.

2. The existential quantifier $\exists$ (there exists or for some). $\exists x\ P(x)$ means there exists x such that $P(x)$ is true if $P(x)$ is true for at least one $x \in U$, otherwise $\exists x\ P(x)$ is false.

   Example: Let U be the set of integers. $\exists x\ [x > 2]$ is true.

   Example: Let U be the set of natural numbers. $\exists x\ [x = 0]$ is false.

   Note that if $P(c)$ is true, then $\exists x\ P(x)$ is true.

3. The unique existential quantifier $\exists!$ (one and only one). $\exists! x\ P(x)$ is true if $P(x)$ is true for exactly one element in U.

   Example: Let U be the set of natural numbers. $\exists! x\ [x < 1]$ is false.

Example: Let U be the set of natural numbers. $\exists! x\ [x = 3]$ is true.

Example: Let U be the set of natural numbers. $\exists! x\ [x < 2]$ is true.

Note: Let $P(x)$ represent $x \leq 5$ where U is the set of natural numbers. Then, $\forall x\ P(x) = P(1) \wedge P(2) \wedge P(3) \wedge P(5) \wedge P(6) \cdots$.

In general, $\forall x\ P(x) = P(x_0) \wedge P(x_1) \wedge \cdots$.

all	at least one	uniqueness
$\forall x\ P(x)$	$\exists x\ P(x)$	$\exists! x\ P(x)$
$\wedge$	$\vee$	$P(x_0) \oplus P(x_1) \oplus P(x_2) \cdots$

When combining more than one quantifier, order is important in predicates with more than one quantifier. The notation $\forall x \forall y\ P(x, y)$ means $\forall x[\forall y\ P(x, y)]$.

Example: $\forall x[\exists y\ P(x, y)]$: for every x there is some y — not necessarilary the same y.

Example: $\exists y[\forall x\ P(x, y)]$: there is some y such that for every x there is a value which makes P true.

Example: $\forall x \forall y\ P(x, y) = \forall y \forall x\ P(x, y)$.

Example: $\exists x \exists y\ P(x, y) = \exists y \exists x\ P(x, y)$.

Read the examples on page 26 in the text book. In Section 1.2 on page 27, do problems 1, 3, 5(d*), 7, and 9.

1.1.6 Quiz 1

1. Using truth tables, show that an implication is equivalent to its contrapositive. Solution:

P	Q	$P \Rightarrow Q$	$\neg Q$	$\neg P$	$\neg Q \Rightarrow \neg P$	$P \Rightarrow Q \Leftrightarrow \neg Q \Rightarrow \neg P)$
0	0	1	1	1	1	1
0	1	1	0	1	1	1
1	0	0	1	0	0	1
1	1	1	0	0	1	1

2. Make a truth table for $(\neg Q \wedge P) \vee (\neg P \Rightarrow Q)$. Is this proposition a tautology, contingency, or contradiction? Solution: contingency.

P	Q	$\neg Q$	$\neg Q \wedge P$	$\neg P$	$\neg P \Rightarrow Q$	$(\neg Q \wedge P) \vee (\neg P \Rightarrow Q)$
0	0	1	0	1	0	0
0	1	0	0	1	1	1
1	0	1	1	0	1	1
1	1	0	0	0	1	1

3. Le P : dogs bark. Q : cats bite. R : horses swim. Write each of the following in symbolic form.

   (a) Horses swim if and only if cats do not bite. Solution: $R \Leftrightarrow \neg Q$.

   (b) If horses swim then dogs do not bark, and if dogs do not bark then cats bite. Solution: $(R \rightarrow \neg P) \wedge (\neg P \Rightarrow Q)$.

   (c) Either dogs bark or cats bite, but not both. Solution: $(P \vee Q) \wedge \neg(P \wedge Q)$.

   (d) It is not the case that horses do not swim and dogs do not bark. Solution: $\neg(\neg R \wedge \neg P)$.

1.1.7 Quantifiers with Negation

This section covers quantifiers with negation.

1. $\neg\forall x\ P(x)$ means $\exists x\neg\ P(x)$. Proof:

   $$\neg(P(x_0) \wedge P(x_1) \wedge P(x_2) \wedge \cdots) =$$

   $$\neg P(x_0) \vee \neg P(x_1) \vee \neg P(x_2) \vee \cdots =$$

   $$\exists x\neg\ P(x).$$

   $\neg\exists x\ P(x)$ means $\forall x\ \neg P(x)$. Proof:

   $$\neg(P(x_0) \vee P(x_1) \vee P(x_2) \vee \cdots) =$$

   $$\neg P(x_0) \wedge \neg P(x_1) \wedge \neg P(x_2) \wedge \cdots =$$

   $$\forall x \neg P(x).$$

   $\neg\forall x\exists y\forall z\ P(x, y, z) = \exists x\forall y\exists z\neg P(x, y, z)$.

2. Conjunction: $\forall x[P(x) \wedge Q(x)] \Leftrightarrow \forall x\ P(x) \wedge \forall x\ Q(x)$. Proof:

   $$[P(x_0) \wedge Q(x_0)] \wedge [P(x_1) \wedge Q(x_1)] \wedge [P(x_2) \wedge Q(x_2)] \wedge \cdots$$

   Regroup.

   $$[P(x_0) \wedge P(x_1) \wedge P(x_2) \wedge \cdots] \wedge [Q(x_0) \wedge Q(x_1) \wedge Q(x_2) \wedge \cdots] =$$

   $$\forall x\ P(x) \wedge \forall x\ Q(x).$$

Here are some properties:

1. $\neg\forall x\ P(x) \Leftrightarrow \exists x\ \neg P(x)$.
2. $\neg\exists x\ P(x) \Leftrightarrow \forall x\ \neg P(x)$.
3. $\forall x P(x) \wedge Q(x)) \Leftrightarrow \forall x\ P(x) \wedge \forall x\ Q(x)$. This is not the same as $\exists x(P(x) \wedge Q(x)) \neq \exists x\ P(x) \wedge \exists x\ Q(x)$. Left-proof:

   $$(P(x_0) \wedge Q(x_0)) \vee (P(x_1) \wedge Q(x_1)) \vee (P(x_2) \wedge Q(x_2)) \vee \cdots .$$

   $$\exists x\ P(x) \wedge \exists x\ Q(x) \Leftrightarrow$$

   Right-proof:

   $$[P(x_0) \vee P(x_1) \vee P(x_2) \vee \cdots] \wedge [Q(x_0) \vee Q(x_1) \vee Q(x_2) \vee \cdots].$$

4. $\exists x(P(x) \wedge Q(x)) \Rightarrow \exists x\ P(x) \wedge \exists x\ Q(x)$ is true.
5. $\forall x(P(x) \vee Q(x)) \neq \forall x\ P(x) \vee \forall x\ Q(x)$. Also, $\exists x(P(x) \vee Q(x)) \Leftrightarrow \exists x\ P(x) \vee \exists x\ Q(x)$.
6. $\forall x(P(x) \vee Q(x)) \Leftarrow \forall x\ P(x) \vee \forall x\ Q(x)$.

The *scope* of a quantifier is that part of an assertion in which variables are bound by the quantifier. It is the smallest subexpression which is consistent with the parenthesis of the expression.

Example: $\forall x[P(x) \wedge Q]$. The scope of $\forall$ is $[P(x) \wedge Q]$.

Example: $\forall x P(x) \wedge Q$. The scope of $\forall$ is $P(x)$.

Example: $\forall x P(x) \wedge Q(x) \Leftrightarrow \forall y P(y) \wedge Q(x)$. The scope of $\forall$ is $P(x)$.

Result: Predicates which occur in $\vee$ and $\wedge$ assertions and which are not bound by a quantifier may be removed from the scope of the quantifier.

Example: $\forall x[P(x) \wedge Q] \Leftrightarrow \forall x\ P(x) \wedge Q$. Q is not bound by $\forall$. Proof:

$$\forall x[P(x) \wedge Q] \Leftrightarrow$$

$$[P(x_0) \wedge Q] \wedge [P(x_1) \wedge Q] \wedge [P(x_2) \wedge Q] \wedge \cdots$$

$$[P(x_0) \wedge P(x_1) \wedge P(x_2) \wedge \cdots] \wedge [Q \wedge Q \wedge Q \wedge \cdots]$$

$$\forall x\ P(x) \wedge Q.$$

Example: $\forall x[P(x) \wedge Q] \Leftrightarrow \forall x\ P(x) \vee Q$. Proof:

$$[P(x_0) \vee Q] \wedge [P(x_1) \vee Q] \wedge [P(x_2) \vee Q] \wedge \cdots$$

$$[P(x_0) \wedge P(x_1) \wedge P(x_2) \wedge \cdots] \vee Q$$

$$\forall x\ P(x) \vee Q.$$

Example: $\exists x[P(x) \wedge Q] \Leftrightarrow \exists x\ P(x) \wedge Q$.

Example: $\exists x[P(x) \vee Q] \Leftrightarrow \exists x\ P(x) \vee Q$.

Example: $\forall x[P(x) \wedge Q(y)] \Leftrightarrow \forall x\ P(x) \wedge Q(y)$.

In Section 1.3 of the textbook, page 37, do problems 1a-d, 2, 3a-c, 4, 9. On Friday, Quiz 2 which will cover Sections 1.2 and 1.3.

1.1.8 Homework and Answers

Section 1.2 homework and answers on Page 27 of the textbook.

1. Let $S(x, y, z)$ denote the predicate $x + y - z$, $P(x, y, z)$ denote $x \cdot y = z$, and $L(x, y)$ denote $x < y$. Let the universe of discourse be the natural numbers N. Using the above predicates, express the following assertions. The phrase "there is an x" does not imply that x has a unique value.

a. For every x and y, there is a z such that $x + y = z$. Solution: $\forall x \forall y \; S(x, y, z)$.

b. No x is less than 0. Solution: $\forall x \neg L(x, 0)$.

c. For all x, $x + 0 = x$. Solution: $\forall x \; S(x, 0, 0)$.

d. For all x, $x \cdot y = y$ for all y. Solution: $\forall x \forall y \; P(x, y, y)$.

e. There is an x such that $x \cdot y = y$ for all y. Solution: $\exists x \forall y \; P(x, y, y)$.

3. Determine which of the following propositions are true if the universe is the set of integers I and $\cdot$ denotes the operation of multiplication.

a. $\forall x, \exists y[x \cdot y = 0]$. True

b. $\forall x, \exists! y[x \cdot y = 1]$. False ($x = 0$).

c. $\exists y, \forall x[x \cdot y = 1]$. False.

d. $\exists y, \forall x[x \cdot y = x]$. True.

5. Specify a universe discourse for which the following propositions are true. Try to choose the universe to be as large a subset of the integers as possible. Explain any difficulties.

a. $\forall x[x > 10]$. Solution: $U = \{11, 12, 13, ...\}$.

b. $\forall x[x = 3]$. Solution: $U = \{3\}$.

c. $\forall x \exists y[x + y = 436]$. Solution: $U =$ integers.

d. $\exists y \forall x[x + y < 0]$. Solution: $U =$ integers, $|y| > |x|$.

7. Consider the univers of integers I.

a. Find the predicate $P(x)$ which is false regardless of whether the variable x is bound by $\forall$ or $\exists$. Solution: $x = x + 1$. $P(x)[x = \frac{1}{2}]$.

b. Find the predicate $P(x)$ which is true regardless of whether the variable x is bound by $\forall$ or $\exists$. Solution: $x = x$. $[x + 1 > x]$.

c. Is it possible for a predicate $P(x)$ to be true regardless of whether the variable is bound by $\forall, \exists$ or $\exists!$? Justify your answer. Solution: Yes. The universal set equal to one element say 1 [$x = 1$ for all cases].

9. Consider the universe of integers and let $P(x, y, z)$ denote $x - y = z$. Transcribe the following assertions into logical notation.

a. For every x and y, there is some z such that $x - y = z$. Solution: $\forall x \forall y \exists z \; P(x, y, z)$.

b. For every x and y, there is some z such that $x - z = y$. Solution: $\forall x \forall z \exists y \; P(x, y, z)$.

c. There is an x such that for all y, $y - x = y$. Solution: $\exists x \forall y \; P(x, y, y)$.

d. When 0 is subtracted from any integer, the result is the original integer. Solution: $\forall x \exists y \; P(x, y, x)$.

e. 3 subtracted from 5 gives 2. Solution: $P(5, 3, 2)$.

Some additional observations: If $\exists P(x)$ is false, then $\forall x \; P(x)$ is false. If $\forall x \; P(x)$ is true, then $\exists x \; P(x)$ is true.

1.1.9 Homework and Answers

This is Section 1.3 homework and answers on page 37 from the textbook.

1. Let $P(x, y, z)$ denote $xy = z$. Let $E(x, y)$ denote $x = y$. Let $G(x, y)$ denote $x > y$. Let the universe of discourse be the integers. Transcribe the following into logical notation.

a. If $y = 1$, then $xy = x$ for any x. Solution: $\forall y[E(y, 1) \Rightarrow \forall x\ P(x, y, x)]$.

b. If $xy \neq 0$, then $x \neq 0$ and $y \neq 0$. Solution: $\forall x \forall y[\neg P(x, y, 0) \Rightarrow \neg E(x, 0) \wedge \neg E(y, 0)]$.

c. If $xy = 0$, then $x = 0$ or $y = 0$. Solution: $\forall x \forall y[P(x, y, 0) \Rightarrow E(x, 0) \vee E(y, 0)]$.

d. $3x = 6$ if and only if $x = 2$. Solution: $\forall x[P(3, x, 6) \Leftrightarrow E(x, 2)]$.

2. Let the universe of discourse be the set of arithmetic assertions with predicates defined as follows: $P(x)$ denotes "x is provable. $T(x)$ denotes "x is true." $S(x)$ denotes "x is satisfiable." $D(x, y, z)$ denotes "z is the disjuntion $x \vee y$." Translate the following assertions into English statements. Make your transcriptions as natural as possible.

a. $\forall x[P(x) \Rightarrow T(x)]$. Solution: For all arithmetic assertions x such that if x is provable, then x is true.

b. $\forall x[T(x) \vee \neg S(x)]$. Solution: For all x such that x is true or x is not satisiable.

c. $\exists x[T(x) \wedge \neg P(x)]$. Solution: For all x such that if x is true, then x is not provable.

d. $\forall x \forall y \forall z[[D(x, y, z) \wedge P(z)] \Rightarrow [P(x) \vee P(y)]]$. Solution: For all x, y, and z if z is disjunction $x \vee y$ and z is provable, then x is provable or y is provable.

e. $\forall x[T(x) \Rightarrow \forall y \forall z[D(x, y, z) \Rightarrow T(z)]]$. Solution: For all x if x is true, then for all y and z if z is the disjunction $x \vee y$ then z is true.

3. Put the following into logical notation. Choose predicates so that each assertion requires at least one quantifier.

a. There is one and only one even prime. Solution: $P(x) =$ the even numbers. $Q(x) =$ the prime numbers. $\exists! x\ [P(x) \wedge Q(x)]$.

b. No odd numbers are even. Solution: $P(x) =$ the even numbers. $Q(x) =$ the odd numbers. $\forall x\ [Q(x) \Rightarrow \neg P(x)]$. $\forall x[\neg P(x) \vee \neg Q(x)]$.

c. Every train is faster than some cars. Solution: $P(x) =$ trains. $Q(y) =$ cars. $R(x, y) =$ trains travel faster than cars. $\forall x[P(x) \Rightarrow \exists y[Q(y) \wedge R(x, y)]$.

d. Some cars are slower than all trains but at least one train is faster than every car. Solution: $\exists x \neg P(x)$. $\neg \forall x\ P(x)$.

9. Show that the following are valid for the universe of natural numbers N either by expanding the statement or by applying identities.

a. $\forall x \forall y[P(x) \vee Q(y)] \Leftrightarrow [\forall x P(x) \vee \forall y Q(y)]$. Solution: $\forall x \forall y[P(x) \vee Q(y)] \Leftrightarrow [\forall x\ P(x) \vee \forall y\ Q(y)]$.

$\forall x[\forall y[P(x) \vee Q(y)] \Leftrightarrow$

$\forall x[P(x) \vee \forall y Q(y)] \Leftrightarrow$

$\forall x P(x) \vee \forall y Q(y)$.

b. $\exists x \exists y[P(x) \wedge Q(y)] \Rightarrow \exists x P(x)$. Solution: $P \wedge Q \Rightarrow P$.

e. $\forall x \forall y[P(x) \Rightarrow Q(y)] \Leftrightarrow [\exists x P(x) \Rightarrow \forall y Q(y)]$. Solution:

$$\forall x \forall y[P(x) \Rightarrow Q(y)] \Leftrightarrow$$

$$\forall x \forall y[\neg P(x) \vee Q(y)]$$

$$\forall x \neg P(x) \vee \forall y Q(y)$$

$$\neg \exists x P(x) \vee \forall y Q(y)$$

$$\exists x P(x) \Rightarrow \forall y Q(y).$$

1.1.10 Quiz 2

September 11, 1987

1. Let the universe be the set of integers. Determine whether each of the following propositions is True or False. Explain your answers.

   (a) $\forall x \exists! y[3x - y = 5]$. Solution: True because if x is positive, y can be negative. If x is negative, y can be positive to make up the difference.

   (b) $\exists x[x = \frac{1}{x}]$. Solution: True at $x = 1$. $\exists$ only takes at least one case to make it true.

   (c) $\forall x \exists y[x = 2y]$. Solution: False. What if x is an odd number? Then, y would have to be a real number that is not in the universe.

   (d) $\forall x \forall y[(x + y > 0) \vee (x + y < 0)]$. Solution: False. What if x and y equal to zero? Then neither case would hold true.

   (e) $\exists y \forall x[y \cdot x = x]$. Solution: True for the case $y = 1$. Then, any x would hold true.

2. Prove the following identity by expanding the left-hand side. $\forall x \neg P(x) \Leftrightarrow \neg \exists x P(x)$. Solution:
   $\neg P(x_0) \wedge \neg P(x_1) \wedge \neg P(x_2) \wedge \cdots \Leftrightarrow$
   $\neg(P(x_0) \vee P(x_1) \vee P(x_2) \vee \cdots) \Leftrightarrow$
   $\neg \exists x P(x)$.

3. Prove the following identity by expanding the left-hand side. $\exists x[Q(y) \wedge P(x)] \Leftrightarrow Q(y) \wedge \exists x P(x)$. Solution:
   $(Q(y) \wedge P(x_0)) \vee (Q(y) \wedge P(x_1)) \vee (Q(y) \wedge P(x_2)) \vee \cdots) \Leftrightarrow$
   $Q(y) \wedge (P(x_0) \vee P(x_1) \vee P(x_2) \vee \cdots) \Leftrightarrow$
   $Q(y) \wedge \exists x P(x)$

1.1.11 Logical Inferences

A *proof* is a sequence of statements which establishes that a theorem is true. The sequence of assertions in a proof consists of three things:

1. Axioms or previously proved theorems.
2. Hypotheses of the theorem.
3. Assertions inferred from previous assertions in the proof.

Some rules of inference include:

1. Modus ponens

$$\begin{array}{l} P \\ P \Rightarrow Q \\ \hline \therefore Q \end{array}$$

Example:

$$\begin{array}{l} \text{If it rains, then I will study.} \\ \text{It rains.} \\ \hline \therefore \text{I will study.} \end{array}$$

An argument iff, the conjunction of the hypotheses, implies the conclusion is a tautology.

P	Q	$P \Rightarrow Q$	$P \wedge (P \Rightarrow Q)$	$[P \wedge (P \Rightarrow Q)] \Rightarrow Q$
0	0	1	0	1
0	1	1	0	1
1	0	0	0	1
1	1	1	1	1

2. Modus tollens.

$$\begin{array}{l} \neg Q \\ P \Rightarrow Q \\ \hline \therefore \neg P \end{array}$$

3. Addition.

$$\begin{array}{l} P \\ \hline \therefore P \vee Q \end{array}$$

4. Simplification.

$$\begin{array}{l} P \wedge Q \\ \hline \therefore P \end{array}$$

5. Disjunctive syllogism.

$$\begin{array}{l} P \vee Q \\ \neg P \\ \hline \therefore Q \end{array}$$

$[(P \vee Q) \wedge \neg P] \Rightarrow Q$.

6. Hypothetical syllogism.

$$\begin{array}{l} P \Rightarrow Q \\ Q \Rightarrow R \\ \hline \therefore P \Rightarrow R \end{array}$$

7. Conjunction.

$$\begin{array}{l} P \\ Q \\ \hline \therefore P \wedge Q \end{array}$$

8. Constructive delimma.

$$\begin{array}{l} (P \Rightarrow Q) \wedge (R \Rightarrow S) \\ P \vee R \\ \hline \therefore Q \vee S \end{array}$$

9. Destructive delimma.

$$\begin{array}{l} (P \Rightarrow Q) \wedge (R \Rightarrow S) \\ \neg Q \vee \neg S \\ \hline \therefore \neg P \vee \neg R \end{array}$$

Some fallacious agruments include the following:

$$\begin{array}{l} P \Rightarrow Q \\ Q \\ \hline \therefore P \end{array}$$

and

$$\begin{array}{l} P \Rightarrow Q \\ \neg P \\ \hline \therefore \neg Q \end{array}$$

Some more complex arguments:

Example: Prove that the following argument is valid.

$$\begin{array}{l} 1. P \Rightarrow Q \\ 2. Q \Rightarrow R \\ 3. \neg R \\ \hline \therefore \neg P \end{array}$$

	Assertions	Reasons
1.	$P \Rightarrow Q$	Hypothesis 1
2.	$Q \Rightarrow R$	Hypothesis 2
3.	$P \Rightarrow R$	Hypotheses 1,2 and hypothetical syllogism
4.	$\neg R$	Hypothesis 3
5.	$\neg P$	Hypotheses 3,4 and modus tollens

Exam #1 will be on Monday, September 21. It covers Sections 1.1 thru 1.4.

Example:

1.$R \Rightarrow \neg S$
2.$\neg M \Rightarrow T$
3.$\neg P \Rightarrow R$
4.$T \Rightarrow S$
$\therefore \neg P \Rightarrow M$

	Assertions	Reasons
1.	$\neg P \Rightarrow R$	Hypothesis 3
2.	$R \Rightarrow \neg S$	Hypothesis 1
3.	$\neg S \Rightarrow \neg T$	Contrapositive
4.	$\neg T \Rightarrow M$	Contrapositive of hypothesis 2
5.	$\neg P \Rightarrow M$	Hypothesis 1,23,4 and hypothetical syllogism

Example: Premises:

1.$A \Rightarrow \neg B$
2.$\neg C \Rightarrow F$
3.$A \vee \neg C$
4.$\neg B \Rightarrow D$
5.$F \Rightarrow H$
$\therefore D \vee H$

	Assertions	Reasons
1.	$A \Rightarrow \neg B$	Hypothesis 1
2.	$\neg B \Rightarrow D$	Hypothesis 4
3.	$A \Rightarrow D$	Hypothesis 1,2
4.	$\neg C \Rightarrow F$	Hypothetical syllogism
5.	$F \Rightarrow H$	Hypotheses 2,5
6.	$\neg C \Rightarrow H$	Hypotheses 4,5 and chain rule
7.	$(A \Rightarrow D) \wedge (\neg C \Rightarrow H)$	Hypotheses 3,6 and conjunction
8.	$A \vee \neg C$	Hypothesis 3
9.	$D \vee H$	Hypotheses 7,8 and constructive delimma

The test covers Sections 1.1 to 1.4, truth tables, the operators $+, -, \oplus, \Rightarrow$, and $\Leftrightarrow$. You must know how to read symbolic form to English and vise-versa; page 15 Table of Identities (omit 20 and 21); page 16 implications (omit 7, 8, and 9); qualifiers — read, figure true / false of propositions; properties; proofs of valid arguments with truth tables; rules of inference (page 41).

1.1.12 Handout

3.

$S \Rightarrow L$	SON $\Rightarrow \neg L$
$\neg S \Rightarrow \neg F$	$\neg L \Rightarrow \neg S$
SON $\Rightarrow \neg L$	$\neg S \Rightarrow \neg F$
$\therefore$	$\therefore$ SON $\Rightarrow \neg F$

4.

$D \Rightarrow \neg W$	$P \Rightarrow D$
$O \Rightarrow W$	$D \Rightarrow \neg W$
$P \Rightarrow D$	$\neg W \Rightarrow \neg O$
	$\therefore p \rightarrow \neg O$

5.

$$\begin{array}{ll} B \Rightarrow I & B \Rightarrow I \\ C \Rightarrow \neg D & I \Rightarrow \neg D \\ I \Rightarrow D & D \Rightarrow \neg C \\ \hline \therefore & \therefore B \Rightarrow \neg C \end{array}$$

1.1.13 Homework Handout Questions and Answers

Verify that the following argument forms are valid.

1. Form:

$$\begin{array}{l} \neg Q \Rightarrow \neg R \\ R \\ P \Rightarrow \neg Q \\ \hline \therefore \neg P \end{array}$$

Solution:

	Assertions	Reasons
1.	$\neg Q \Rightarrow \neg R$	Hypothesis 1
2.	$R \Rightarrow Q$	Assertions 1, Contrapositive
3.	R	Hypothesis 2
4.	$Q \Rightarrow \neg P$	Hypothesis 3, Contrapositive
5.	Q	Assertions 3, 2, Modus ponens
6.	$\neg P$	Assertions 4, 5, Modus ponens

2. Form:

$$\begin{array}{l} (P \Rightarrow \neg Q) \wedge (\neg R \Rightarrow S) \\ \neg P \Rightarrow \neg R \\ \hline \therefore \neg Q \vee S \end{array}$$

Solution:

	Assertions	Reasons
1.	$\neg P \Rightarrow \neg R$	Hypothesis 2
2.	$P \vee \neg R$	Equivalence
3.	$(P \Rightarrow \neg Q) \wedge (\neg R \Rightarrow S)$	Hypothesis 1
4.	$\neg Q \vee S$	Assertions 2,3, Constructive dilemma

3. Form:

$$\begin{array}{l} \neg B \Rightarrow F \\ I \Rightarrow \neg C \\ A \Rightarrow \neg B \\ F \Rightarrow I \\ \hline \therefore A \Rightarrow \neg C \end{array}$$

Solution:

	Assertions	Reasons
1.	$A \Rightarrow \neg B$	Hypothesis 3
2.	$\neg B \Rightarrow F$	Hypothesis 1
3.	$F \Rightarrow I$	Hypothesis 4
4.	$I \Rightarrow \neg C$	Hypothesis 2
5.	$A \Rightarrow \neg C$	Assertions 1,2,3,4, Hypothetical syllogism

4. Form:

$$\begin{array}{l} N \Rightarrow \neg R \\ \neg M \Rightarrow N \\ R \\ P \\ \hline \therefore M \wedge P \end{array}$$

Solution:

	Assertions	Reasons
1.	$N \Rightarrow \neg R$	Hypothesis 1
2.	$R \Rightarrow \neg N$	Hypothesis 1, Contrapositive
3.	$\neg M \Rightarrow N$	Hypothesis 2
4.	$\neg N \Rightarrow M$	Assertion 3, Contrapositive
5.	$R \Rightarrow M$	Assertions 2,4, Hypothetical syllogism
6.	R	Hypothesis 3
7.	M	Assertion 5, 6, Modus ponens
8.	P	Hypothesis 4
9.	$M \wedge P$	Assertion 7, 8, Conjunction

5. Form:

$$\begin{array}{l} \neg F \Rightarrow C \\ \neg C \vee \neg D \\ B \Rightarrow \neg F \\ D \\ \hline \therefore \neg B \end{array}$$

Solution:

	Assertions	Reasons
1.	$\neg C \Rightarrow \neg D$	Hypothesis 2
2.	$C \Rightarrow \neg D$	Hypothesis 2, Equivalence
3.	$D \Rightarrow \neg C$	Assertion 2, Contrapositive
4.	$\neg C \Rightarrow F$	Hypothesis 1, Contrapositive
5.	$F \Rightarrow \neg B$	Hypothesis 3, Contrapositive
6.	D	Hypothesis 4
7.	$\neg B$	Assertions 3, 4, 5, 6, Hypothetical syllogism

1.1.14 Methods of Proof

There are several techniques for proving implications.

1. Vacuous proof of $P \Rightarrow Q$. (Rows 1, 2) Recall $P \Rightarrow Q$ is *True* whenever P is *False.* If we can establish that P is *False,* then we say vacously that $P \Rightarrow Q$ is *True.*

2. Trivial proof of $P \Rightarrow Q$ (Rows 2,4) Observe that $P \Rightarrow Q$ is true whenever Q is *True.* So, if we establish Q as *True,* then $P \Rightarrow Q$ is *True.*

3. Direct proof of $P \Rightarrow Q$. $P \Rightarrow Q$ is *True* when P and Q are both *True.* Assume that P is *True* and show that Q is also *True.*

4. Indirect proof of $P \Rightarrow Q$. We show that the contrapositive is true by direct proof. Assume $\neg Q$ is *True* and show that $\neg P$ is also True.

5. If the premise is a conjunction $(P_0 \wedge P_1 \wedge P_2 \wedge P_3 \cdots \wedge P_n) \Rightarrow Q$, we look at the contrapositive $\neg Q \Rightarrow \neg(P_1 \wedge P_2 \wedge P_3 \wedge \cdots P_n)$ or $\neg Q \Rightarrow \neg P_1 \vee \neg P_2 \vee \neg P_3 \vee \cdots \neg P_n$ is *True* if $\neg Q \Rightarrow \neg P_i$ for at least one i.

6. If the premise is a disjunction,

$$(P_0 \vee P_2 \vee P_3 \vee \cdots \vee P_n) \Rightarrow Q \Leftrightarrow$$

$$\neg(P_0 \vee P_2 \vee P_3 \vee \cdots \vee P_n) \vee Q \Leftrightarrow$$

$$(\neg P_0 \wedge \neg P_2 \wedge \neg P_3 \wedge \cdots \wedge \neg P_n) \vee Q \Leftrightarrow$$

$$(\neg P_0 \vee Q) \wedge (\neg P_2 \vee Q) \wedge (\neg P_3 \vee Q) \wedge \cdots \wedge (\neg P_n \vee Q) \Leftrightarrow$$

$$(P_0 \Rightarrow Q) \wedge (P_2 \Rightarrow Q) \wedge \cdots \wedge (P_n \Rightarrow Q)$$

is *True* when $P_i \Rightarrow Q$ for all i between 0 and n called a *proof by cases.*

Some other proof techniques include:

1. Proofs of equivalence.
   (a) Use $P \Leftrightarrow Q$ means $(P \Rightarrow Q)$ and $(Q \Rightarrow P)$. Show the two parts seperately. $P \Leftrightarrow Q$ is also read if and only if.
   (b) Begin with an equivalence $R \Leftrightarrow S$ and proceed through a sequence of equivalencies to eventually generate $P \Leftrightarrow Q$.

2. Proof by contradiction. Assume that the opposite with negation is true. Eventually, you arrive at a contridiction. The contradiction implies that the assumption was incorrect.
   Note: Theorem: $P \Rightarrow Q$. Assume $\neg(P \Rightarrow Q)$ is true. Then,

$$\neg(P \Rightarrow Q)$$

$$\neg(\neg P \vee Q)$$

$$(P \wedge \neg Q)$$

   i.e. assume *True* so that P is *True* and Q is *False.* $0 = 1$ is a contradiction to $\neg(P \Rightarrow Q) \therefore P \Rightarrow Q$.

Theorem: For all integers x, x is even iff x^2 is even. Proof: Case 1 (the only if part). Show if x is even, then x^2 is even. $x^2 = (2k)^2 = 4k^2 = 2(2k^2)$ where $2k^2 \in I$. So, x^2 is even. Case 2 (the if part). Show if x^2 is even, then x is even. Use an indirect proof. Prove if x is not even, then x^2 is not even. x is not even $\Rightarrow x$ is odd $\Rightarrow x = 2k + 1$ where $k \in I \Rightarrow x^2 = (2k + 1)^2 \Rightarrow x^2 = 4k^2 + 4k + 1 \Rightarrow x^2 = 2(2k^2 + 2k) + 1 \in I \therefore x^2$ is odd (i.e. x^2 is not even).

Theorem: If the product of two integers (A and B) is even, then either A is even or B is even. $A \times B$ even $\Rightarrow A$ even $\vee B$ even. Proof (indirect proof):

$$\neg(A \text{ even } \vee B \text{ even }) \Rightarrow \neg AB \text{ even}$$

Show

A odd $\wedge\, B$ odd $\Rightarrow AB$ odd

A odd $\wedge\, B$ odd $\Rightarrow A = 2k+1 \wedge B = 2n+1 \;\; k, n \in I$

$\Rightarrow AB = (2k+1)(2n+1)$

$\Rightarrow AB = 4kn + 2k + 2n + 1 =$

$2(2kn + k + n) + 1,$

$2kn + k + n \in I$

$\therefore AB$ is odd.

Theorem: If A is an integer, such that $A-2$ is divisible by 3, then $A^2 - 1 = (A+1)(A-1)$ is divisible by 3. Proof (direct proof): Assume that $A-2$ is divisible by $3 \Rightarrow A-2 = 3k$ where $k \in I \Rightarrow A-2+3 = 3k+3$ $\Rightarrow A+1 = 3(k+1) \Rightarrow A^2 - 1 = (A+1)(A-1) = 3(k+1)(k-1)$. $(k+1), (k-1) \in I \therefore A^2 - 1$ is divisible by 3.

Theorem: If n is a prime number different from 2, then n is odd. A prime number is any natural number greater than 1 which is divisible only by itself and 1.

Proof: By contradiction. Recall that to prove $P \Rightarrow Q$, assume $\neg(P \Rightarrow Q) \Leftrightarrow (P \wedge \neg Q)$. Assume n is prime and $n \neq 2$ and n is even.
n is even
$\Rightarrow n = 2k$ where $k \in$ Interger, $k \neq 1$.
$\Rightarrow n$ is divisible by 2
n is not prime
Contradiction: n is prime.
$\therefore$ If n is prime and $\neq 2$, then n is odd.

Homework: page 56, 1a, b, c, i, j, m.

1.1.15 Proof Techniques for Quantified Assertions

1. Assertions of the form $\neg\exists x P(x)$ can be proved best by contradiction. Assume $\exists x P(x)$ and derive a contradiction.
2. To prove $\exists x P(x)$:
   (a) Constructive existence proof — find a c such that $P(c)$ is true $\Rightarrow \exists x P(x)$ is true.
   (b) Non-constructive existence proof — a proof by contradiction. Assume $\neg\exists x P(x)$ and derive a contradiction.
3. To prove $\neg\forall x P(x)$:
   (a) Constructive proof: Find a particular value of x, c such that $P(c)$ is false.
   (b) Non-constructive proof: proof by contradiction.
4. To prove $\forall x P(x)$ show that $P(x)$ is true for an arbitrary x.

Homework: all of # 1 on page 56 in the textbook.

1.1.16 Homework and Answers

Problem 1 on page 56 in the textbook.

1. Prove or disprove each of the following assertions. Indicate the proof technique employed. Consider the universe to be the the set of integers I. Put each assertion into logical notation. You must assume the following five definitions and properties of itegers.

    1. An integer n is even if and only if $n = 2k$ for some integer k.
    2. An integer n is odd if and only if $n = 2k + 1$ for some integer k.
    3. The product of two non-zero integers is positive if and only if the integers have the same sign.
    4. For every pair of integers x and y, exactly one of the following holds: $x > y$, $x = y$ or $x < y$.
    5. If $x > y$, then $x - y$ is positive. If $x = y$ then $x - y = 0$. If $x < y$, then $x - y$ is negative.

a. An integer is odd if its square is odd. Solution: $\forall x[x^2 \text{ odd} \Rightarrow x \text{ odd}]$. If x^2 is odd, then x is odd. Proof (indirect): If x is not odd, then x^2 is not odd.

x is not odd $\Rightarrow x$ is even.
$\Rightarrow x = 2k$ where $k \in$ Integer.
$\Rightarrow x^2 = (2k)^2$.
$\Rightarrow x^2 = 4k^2 = 2(2k^2) \in$ Integer.
$\Rightarrow x^2$ is even.
x^2 is not odd.

b. The sum of two even integers is an even integer. Solution: If two even integers are added, then the result is an even integer. Proof (direct): $\forall x \forall y[x \text{ even} \wedge y \text{ even} \Rightarrow x + y \text{ even}]$
x, y are even $\Rightarrow x = 2k, k \in$ Integers.
$y = 2n, n \in$ Integers.
$\Rightarrow x + y = 2k + 2n$.
$\Rightarrow x + y = 2(k + n) \in I$.
$\therefore x + y$ is even.

c. The sum of an even integer and an odd integer is an odd integer. Solution: If an even and odd number are added together, then the result is odd. $\forall x \forall y[x \text{ even} \wedge y \text{ odd} \Rightarrow x + y \text{ odd}]$.
$x + y$ is odd $\Rightarrow x = 2k$, $k \in$ Integers.
$y = 2n + 1, n \in I$.
$\Rightarrow x + y = 2k + 2n + 1$.
$\Rightarrow x + y = 2(k + n) + 1$.
$\Rightarrow 2(k + n) + 1 \in$ Integer.
$\therefore x + y$ is odd.

d. There are two odd integers whose sum is odd: Solution: $\exists x \exists y[x \text{ odd} \wedge y \text{ odd} \wedge x + y \text{ odd}]$. Show it is a false statement. Show $\neg\exists x \exists y[x \text{ odd} \wedge y \text{ even} \wedge x + y \text{ odd}]$ is true.
$\forall x \forall y \neg[(x \text{ odd} \wedge y \text{ odd}) \wedge x + y \text{ odd}]$.
$\forall x \forall y \neg[(x \text{ odd} \wedge y \text{ odd}) \vee \neg[x + y \text{ odd}]$.
$\forall x \forall y[(x \text{ odd} \wedge y \text{ odd}) \Rightarrow \neg[x + y \text{ odd}]$.
$\forall x \forall y[x \text{ odd} \wedge y \text{ odd} \Rightarrow x + y \text{ even}]$.
x odd $\Rightarrow 2k + 1, k \in I$.
y odd $\Rightarrow 2n + 1, n \in I$.
$x + y = 2k + 2n + 2 = 2(k + n + 1), k + n + 1 \in I$.
$\therefore x + y$ is even.

e. The square of any integer is negative. Solution: $\forall x[x^2 < 0]$ is false. If $x = 1, x^2 = 1 > 0$; $x = 1$ is a counter example.

g. There does not exist an integer x such that $x^2 + 1$ is negative. Solution: $\neg\exists x[x^2 + 1 < 0]$ true.
$\forall x\neg[x^2 + 1 < 0]$.
$\forall x[x^2 + 1 \geq 0]$.
$x^2 = x \times x \geq 0$.
$\Rightarrow x^2 + 1 \geq 0$.

i. If $1 = 3$, then the square of any integer is negative. Solution: Vacuous proof.

j. If $1 = 3$, then the square of any integer is positive. Solution: Vacuous proof.

k. The sum of any two primes is a prime number. Solution: 7, 11.

l. There exists two primes whose sum is prime: Solution: 2,3.

m. If the square of any integer is negative, then $1 = 1$. Solution: Vacuous proof.

1.2 Sets

A *set* is a collection of objects or elements. Capital letters denote sets and lower case letters denote elements of sets. To describe a set:

1. List elements $\{1, 2, 3\}$.
2. Set builder notation $\{x|x$ is a natural number$\}$.
$\{\frac{p}{q}|p$ and q are integers and $q \neq 0\}$ = rational numbers.

The *null set* or the *empty set* is the set which contains no elements. This is denoted by $\emptyset$. The null set $\emptyset \neq \{\emptyset\}$. Sets can be:

1. Finite — has a last element.
2. Infinite

Sets A and B are equal ($A = B$) if they contain exactly the same elements $\{1, 2, 3\} = \{3, 2, 1\}$. $A = B$ iff every element in A is also in B and every element in B is also in A. $A = B$ iff $\forall x[(x \in A \Rightarrow x \in B) \wedge (x \in B \Rightarrow x \in A)]$.

A is a subset of B if every element of A is also an element of B. This is denoted by $A \subset B$. $A \subset B \Leftrightarrow \forall x[x \in A \Rightarrow x \in B]$. If $A \subset B$ then $B \supset A$.

Let U be a universal set. Then, $A \subset U$ for all sets A. Proof: $x \in A \Rightarrow x \in U$. $A \subset U$ by definition.

$(A = B)$ iff $(A \subset B \wedge B \subset A)$. Proof: Only if part (left to right) definition of subsets.

Corrolary: (falls directly from something proved) $A \subset A$. Proof: Since every set is equal to itself, then every set must be a subset of itself.

If $A \subset B$ and $B \subset C$, then $A \subset C$. Proof:
$A \subset B \Leftrightarrow x \in A \Rightarrow x \in B$.
$B \subset C \Leftrightarrow x \in B \Rightarrow x \in C$.
$A \subset C \Leftrightarrow x \in A \Rightarrow x \in C$.

Prove $\emptyset \subset A, \forall A$. Proof: $\emptyset \subset A \Leftrightarrow x \in \emptyset \Rightarrow x \in A$. Vacuous proof. $x \in \emptyset$ has no elements and therefore always false making the implication always true.

Let $\emptyset$ and $\emptyset'$ both be null sets. Then, $\emptyset = \emptyset'$. Proof: if $\emptyset$ is null, then $\emptyset \subset \emptyset'$. If $\emptyset'$ is null then $\emptyset' \subset \emptyset$. $\therefore \emptyset = \emptyset'$.

A *singleton* set is a set with exactly one element. For example, $S = \{a\}$. Result: every singleton set has exactly two subsets, the $\emptyset$ set and the subset itself. Result: $\emptyset$ has one subset, $\emptyset$.

Example: $S = \{a, b\}$. The subsets are $\emptyset, S, \{a\}, \{b\}$. (2^2).

Example: $S = \{a, b, c\}$ has 8 choices (2^3).

Example: $S = \{x_1, x_2, ..., x_n\}$ has 2^n.

A set with n elements has 2^n subsets.

Homework: page 79, 1-4 all, section 2.2 omit; section 2.3 page 84 1, 3, 4, 5.

Quiz # 3 — Friday October 9. 1 proof from Section 1.5. 2-3 problems from Sections 2.1 and 2.3.

Exam # 2 — Friday October 16. Section 1.5, 2.1, 2.3, 2.4.

1.2.1 Exam 2

1. Prove each of the following assertions using the method indicated.

   (a) The cube of an even integer is even. Direct proof. Solution: If x is even, then x^3 is even. Assume x is even,
   $\Rightarrow x = 2k$ where k is an integer
   $\Rightarrow x^3 = (2k)^3$
   $\Rightarrow x^3 = 8k^3$
   $\Rightarrow x^3 = 2(4k^3)$ where $4k^3 \in$ Integer
   $\therefore x^3$ is even.

   (b) If x is an odd integer, then x^2 is odd. Proof by contradiction. Solution: $\neg(x$ odd $\Rightarrow x^2$ odd)
   $\Leftrightarrow \neg(\neg x$ odd $\vee x^2$ odd)
   $\Leftrightarrow x$ odd $\wedge \neg x^2$ odd
   $\Leftrightarrow x$ odd $\wedge x^2$ even
   Let $x = 2k + 1$ where $k \in$ Integer
   Then, $x^2 = (2k + 1)^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1$
   where $2k^2 + 2k \in$ Integer
   $\therefore x^2$ is odd which contradicts x^2 even.
   $\therefore x^2$ is even.

   (c) There is a prime number which is not odd. Constructive existence. Solution: Find a prime x such that $x = 2k$. 2 is an even prime number.

   (d) If x^2 is an odd integer, then x is an odd integer. Indirect proof. Solution: if x is not odd, then x^2 is not odd. Assume x is not odd $\Rightarrow x$ is even
   $\Rightarrow x = 2k$ where $k \in$ Integer
   $\Rightarrow x^2 = (2k)^2$
   $\Rightarrow x^2 = 4k^2$
   $\Rightarrow x^2 = 2(2k^2)$ where $2k^2 \in$ Integer

$\therefore x^2$ is even.
$\therefore x^2$ is not odd.

2. Give all subsets of $\{\emptyset, \{1\}, 2\}$. Solution: $2^3 = 8$. $\emptyset$, $\{\emptyset\}$, $\{\{1\}\}$, $\{2\}$, $\{\emptyset, \{1\}\}$, $\{\emptyset, 2\}$, $\{\{1\}, 2\}$, and $\{\emptyset, \{1\}, 2\}$.

3. Draw a Venn diagram for $(A - B) \cup C$. See Figure 1.1.

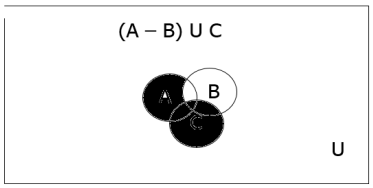

Figure 1.1: Venn diagram for problem 3a on Exam 2.

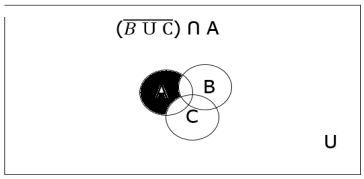

Figure 1.2: Venn diagram for problem 3b on Exam 2.

   (a) Draw a Venn diagram for $\overline{(B \cup C)} \cap A$. See Figure 1.2.

4. Let $U = \{5, 6, 7, ..., 15\}$, $A = \{5, 9, 10, 11, 14\}$, $B = \{5, 7, 8, 9, 11, 15\}$, and $C = \{6, 7, 9, 10, 14, 15\}$. Give each of the following sets.

   (a) $\overline{A} \cap B$. Solution: $\overline{A} = \{6, 7, 8, 12, 13, 15\}$. $\overline{A} \cap B = \{7, 8, 15\}$.
   (b) $C - (A \cup B)$. Solution: $A \cup B = \{5, 7, 8, 9, 10, 11, 14, 15\}$. $C - (A \cup B) = \{6\}$.
   (c) $\overline{(A \cap C)}$. Solution: $A \cap C = \{9, 10, 14\}$. $\overline{(A \cap C)} = \{5, 6, 7, 8, 11, 12, 13, 15\}$.

5. Prove the following identities.

   (a) $A - \emptyset = A$. Solution: Let $x \in A - \emptyset \Leftrightarrow x \in A \wedge x \in \emptyset \Leftrightarrow x \in A \wedge 1 \Leftrightarrow x \in A$. $\therefore A - \emptyset = A$.
   (b) $A \cup (\overline{A} \cap B) = A \cup B$. Solution: Let $x \in A \cup (\overline{A} \cap B)$. Then, $x \in A \vee x \in (\overline{A} \cap B) \Leftrightarrow x \in A \vee (x \notin A \wedge x \in B) \Leftrightarrow (x \in A \vee x \notin A) \wedge (x \in A \vee x \in B) \Leftrightarrow 1 \wedge (x \in A \vee x \in B) \Leftrightarrow x \subset A \vee x \in B \Leftrightarrow x \in A \cup B$ $\therefore A \cup (\overline{A} \cap B) = A \cup B$.

1.2.2 Quiz 3

1. Probe the following assertion by using a direct proof: The product of an even integer and an odd integer is even. Solution:
   x is even $\wedge y$ is odd $\Rightarrow x \cdot y$ even
   x even $\wedge$ y odd $\Rightarrow$
   $x = 2k$ where $k \in I$
   $y = 2n + 1$ where $n \in I$
   $\Rightarrow x \cdot y = 2k(2n + 1)$
   $\Rightarrow x \cdot y = 4kn + 2k$

$\Rightarrow x \cdot y = 2(2kn + k)$
$\Rightarrow (2k + k) \in I$
$\therefore x \cdot y$ even.

2. List all subsets of $\{\{2\}, 5\}$. Solution: $\emptyset$, $\{\{2\}, 5\}$, $\{\{2\}\}$, $\{5\}$.

3. Let $S = \{4, 5, 9, 12, 14, 20\}$. How many subsets does S have? Solution: $2^6 = 64$ subsets.

4. Disprove the following statements by finding a counterexample for each.

   (a) $(A \in B \wedge B \notin C) \Rightarrow A \notin C$. Solution: $A \in B \wedge B \notin C \wedge A \in C$.

   (b) $(A \subset B \wedge A \subset C) \Rightarrow B \subset C$. Solution: $A = \{1\}$, $B = \{1, 2, 3\}$, $C = \{1, 2, 4\}$.

1.2.3 Operations on Sets

The *union* of sets A and B is $A \cup B = \{x | x \in A \vee x \in B\}$. The *intersection* of sets A and B is $A \cap B = \{x | x \in A \wedge x \in B\}$. The difference of sets A and B is $A - B = \{x | x \in A \wedge x \notin B\}$.

Example: $A = \{1, 2, 3, 4, 5\}$. $B = \{2, 4, 6\}$. $A \cup B = \{1, 2, 3, 4, 5, 6\}$. $A \cap B = \{2, 4\}$. $A - B = \{1, 3, 5\}$.

Sets A and B are called *disjoint* if they have no elements in common. $A \cap B = \emptyset$. A collection, C, of sets in which any two elements are disjoint is called a collection of *pairwise disjoint sets.*

Example: $C = \{\{1, 2, 3\}, \{4\}, \{5, 6\}\}$.

The union and intersection are both commutative and associative operations. $A \cup B = B \cup A$ is commutative. $A \cap B = B \cap A$ is commutative. $A \cup (B \cup C) = (A \cup B) \cup C$ is associative and $A \cap (B \cap C) = (A \cap B) \cap C$ is associative. Proof:
Show that $A \cap B = B \cap A$.
$x \in A \cap B \Leftrightarrow x \in A \wedge x \in B$.
$\Leftrightarrow x \in B \wedge x \in A$.
$\Leftrightarrow x \in B \cap A$.

Recall:

1. $R \subset S \Leftrightarrow x \in R \Rightarrow x \in S$.

2. If $R \subset S$ and $S \subset R$, then $R = S$.

The union and intersection distribute over each other. $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ and $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$. Proof:
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
$x \in A \cup (B \cap C) \Leftrightarrow x \in A \vee x \in (B \cap C)$
$\Leftrightarrow x \in A \vee [x \in B \wedge x \in C]$
$\Leftrightarrow (x \in A \vee x \in B) \wedge (x \in A \vee x \in C)$
$\Leftrightarrow x \in (A \vee B) \wedge x \in (A \cup C)$
$\Leftrightarrow x \in (A \cup B) \cap (A \cup C)$
$\therefore A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$

Page 88, Theorem 2.43: Show that $A \cup \emptyset = A$. Proof:
$x \in (A \cup \emptyset) \Leftrightarrow x \in A \vee x \in \emptyset$.
$\Leftrightarrow x \in A \vee \emptyset$.
$\Leftrightarrow x \in A$.
$\therefore A \cup \emptyset = A$.

Theorem: $A - B \subset A$. Proof:
$x \in A - B \Leftrightarrow x \in A \wedge x \notin B$.
$\Rightarrow x \in A$.
$\therefore A - B \subset A$.

Theorem: $A \subset A \cup B$. Proof:
$x \in A \Rightarrow x \in A \vee x \in B$.
$\Leftrightarrow x \in A \cup B$.
$\therefore A \subset A \cup B$.

1.2.4 Homework and Answers

Section 2.1, page 79 1-4 in the textbook.

1. Specify the following sets explicitly.

a. The set of non-negative integers less than 5. Solution: $\{0, 1, 2, 3, 4\}$.

b. The set of letters in your first name. Solution: $\{r, o, g, e, r\}$.

c. The set whose only element is the first president of the United States. Solution: $\{washington\}$.

d. The set of prime numbers between 10 and 20. Solution: $\{11, 13, 17, 19\}$.

e. The set of positive multiples of 12 which are less than 65. Solution: $\{12, 24, 48, 60\}$.

2. For each of the following, choose an appropriate universe of discourse and a predicate to define the set. Do not use ellipses.

a. The set of integers between 0 and 100. Solution: $\{x | x > 0 \wedge x < 100\}$.

b. The set of odd integers. Solution: $\{x | \exists y[x = 2y + 1]\}$.

c. The set of integer multiples of 10. Solution: $\{x | \exists y[x = 10y]\}$.

d. The set of human fathers. Solution: $U =$ all humans. $\{x | x$ is a father $\}$.

e. The set of tautologies. Solution: $U =$ tautalogies.

3. List the members of the following sets.

a. $\{x | x \in I \wedge 3 < x < 12\}$. Solution: $\{4, 5, 6, 7, 8, 9, 10, 11\}$.

c. $\{x | x = 2 \vee x = 5\}$. Solution: $\{2, 5\}$.

4. Determine which of the following sets are equal. The universe of discourse is I. $A = \{x | x$ is even and x^2 is odd$\}$. $B = \{x | \exists y[y \in I \wedge x = 2y]\}$. $C = \{1, 2, 3\}$. $D = \{0, 2, -2, 3, 4, -4, ...\}$. $E = \{2x | x \in I\}$. $F = \{3, 3, 2, 1, 2\}$. $G = \{x | x^3 - 6x^2 - 7x - 6 = 0\}$. Solution: $A = \emptyset$. $B = \{-4, -2, 0, 2, 4\}$. $C = \{1, 2, 3\}$. $D = \{0, 2, -2, 3, -3, 4, -4, ...\}$. $E =$ evens. $F = \{3, 2, 1\}$. $G = \emptyset$. Result $\frac{p}{a}$ where p is a factor of a_0 and a is a factor of a_n. $B = E$, $C = F$, and $A = G$.

1.2.5 Homework and Answers

Page 84: Section 2.3, problems 1, 3, 4, 5 in the textbook.

1. List all alphabets of the following sets.

a. $\{1, 2, 3\}$. Solution: $2^3 = 8$. $\{1\}$, $\{2\}$, $\{3\}$, $\{1, 2\}$, $\{1, 3\}$, $\{2, 3\}$, $\{1, 2, 3\}$, $\emptyset$.

b. $\{1, \{2, 3\}\}$. Solution: 2^2. $\emptyset$, $\{1\}$, $\{\{2, 3\}\}$, $\{1, \{2, 3\}\}$.

c. $\{\{1, \{2, 3\}\}\}$. Solution: $\emptyset$, $\{\{1, \{2, 3\}\}$.

d. $\emptyset$, Solution: $\{\emptyset\}$.

e. $\{\emptyset, \{\emptyset\}\}$, Solution: $\emptyset$, $\{\emptyset\}$, $\{\{\emptyset\}\}$.

3. Let A, B and C bet sets. If $A \in B$ and $B \in C$, is it possible that $A \in C$? Is it always true that $A \in C$? Give examples to support your assertions. Solution: Yes $A = \{1\}$. No $B = \{2, \{5, 6\}, \{1\}\}$. Not always true $C = \{\{1\}, \{2, \{5, 6\}\}\}$. $A \in B$, $B \in C$. $\therefore A \in C$.

5. Briefly describe the differnce between the sets $\{2\}$ and $\{\{2\}\}$. List the elements and all the subsets of each set. Solution: $\{2\}$ contains the element 2. $\{\{2\}\}$ contains the set $\{2\}$. The type of element is different. The set $\{2\}$ has the subsets $\emptyset, \{2\}$. The set $\{\{2\}\}$ has the subset $\{2\}$.

1.2.6 Homework and Answers

Page 88 Theorem 2.4.3 prove parts (m), (n), and (p).

m. Prove $A \cap (B - A) = \emptyset$. Solution:
$x \in A \cap (B - A) \Leftrightarrow$
$x \in A \wedge x \in (B - A) \Leftrightarrow$
$x \in A \wedge (x \in B \wedge x \notin A) \Leftrightarrow$
$(x \in A \wedge x \notin A) \wedge x \in B \Leftrightarrow$
$0 \wedge x \in B \Leftrightarrow$
$0 \Leftrightarrow$
$x \in \emptyset$
$\therefore A \cap (B - A) = \emptyset$.

n. Prove $A \cup (B - A) = A \cup B$. Solution:
$x \in A \cup (B - A) \Leftrightarrow$
$x \in A \vee x \in (B - A) \Leftrightarrow$
$x \in A \vee (x \in B \wedge x \notin A) \Leftrightarrow$
$(x \in A \vee x \in B) \wedge (x \in A \vee x \notin A) \Leftrightarrow$
$(x \in A \vee x \in B) \wedge 1 \Leftrightarrow$
$x \in A \vee x \in B$
$x \in (A \cup B)$.

p. Prove $A - (B \cap C) = (A - B) \cup (A - C)$. Solution:
$x \in A - (B \cap C) \Leftrightarrow$
$x \in A \wedge x \notin (B \cap C) \Leftrightarrow$
$x \in A \wedge \neg x \in (B \cap C) \Leftrightarrow$
$x \in A \wedge \neg(x \in B \wedge x \in C) \Leftrightarrow$
$x \in A \wedge (\neg x \in B \vee \neg x \in C) \Leftrightarrow$
$(x \in A \wedge \neg x \in B) \vee (x \in A \wedge \neg x \in C) \Leftrightarrow$
$(x \in A \wedge x \notin B) \vee (x \in A \wedge x \notin C) \Leftrightarrow$
$(x \in (A - B)) \vee (x \in (A - C)) \Leftrightarrow$
$x \in (A - B) \cup (A - C) \Leftrightarrow$
$\therefore A - (B \cap C) = (A - B) \cup (A - C)$.

1.2.7 Complement of Sets

Let $A \subset U$. Then, the *complement* of A is denoted by $\bar{A}$ and is given by $\bar{A} = U - A = \{x | x \notin A\}$.

Example: Let $U = \{1, 2, 3, 4, 5\}$ and $A = \{2, 5\}$. Then, $\bar{A} = \{1, 3, 4\}$.

Let $A \subset U$. Then the following statements are true:

1. $A \cup \bar{A} = U$.
2. $A \cap \bar{A} = \emptyset$.

Proof: $A \cup \overline{A} = U$.
$x \in A \cup \overline{A} \Leftrightarrow$
$x \in A \vee x \in \overline{A} \Leftrightarrow$
$x \in A \vee x \notin A \Leftrightarrow$
$1 \Leftrightarrow$
$x \in U$.
$\therefore A \cup \overline{A} = U$.

Proof: $A \cap \overline{A} = \emptyset$. Let $A \subset U \wedge B \subset U$. Then, $B = \overline{A}$ iff $A \cup B = U \wedge A \cap B = \emptyset$. Proof (only part): If $B = \overline{A}$, then $A \cup B = A \cup \overline{A} = U \wedge A \cap B = A \cap \overline{A} = \emptyset$. If $A \cup B = U \wedge A \cap B = \emptyset$, then $B = \overline{A}$.
$B = U \cap B$
$= (A \cup \overline{A}) \cap B$
$= (A \cap B) \cup (\overline{A} \cap B)$
$= \emptyset \cup (\overline{A} \cap B)$
$= (\overline{A} \cap A) \cup (\overline{A} \cap B)$
$= \overline{A} \cap (A \cup B)$
$= \overline{A} \cap U$
$= \overline{A}$.

Let $A \subset U$. Then, $\overline{\overline{A}} = A$.

1.2.8 Venn Diagrams

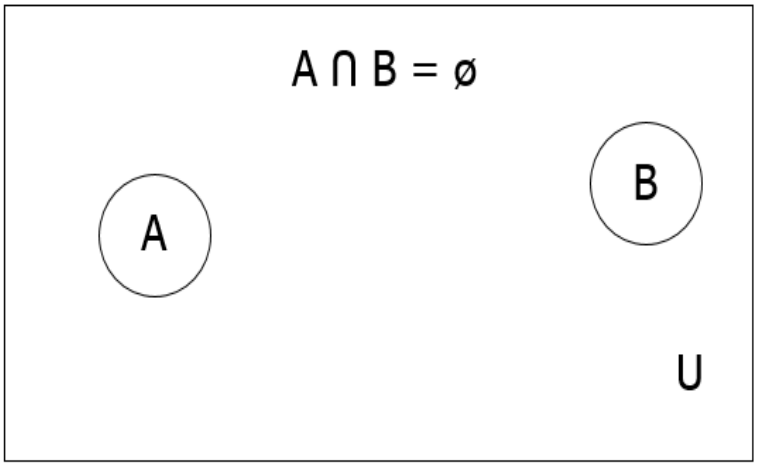

Figure 1.3: Venn diagram of the intersection of two sets being the null value.

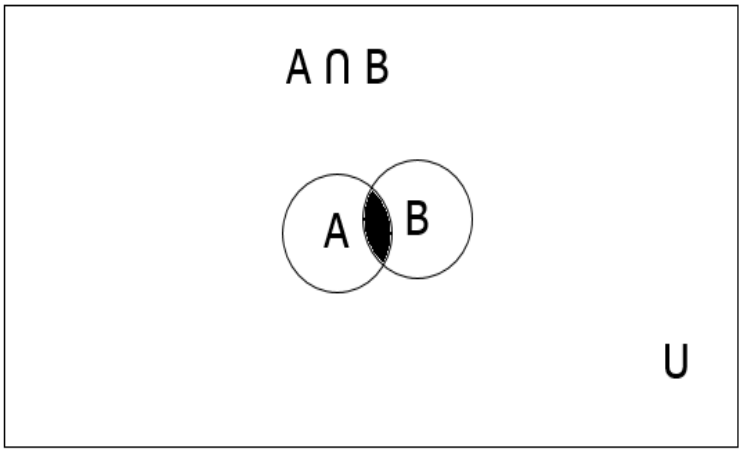

Figure 1.4: Venn diagram of the intersection of two sets.

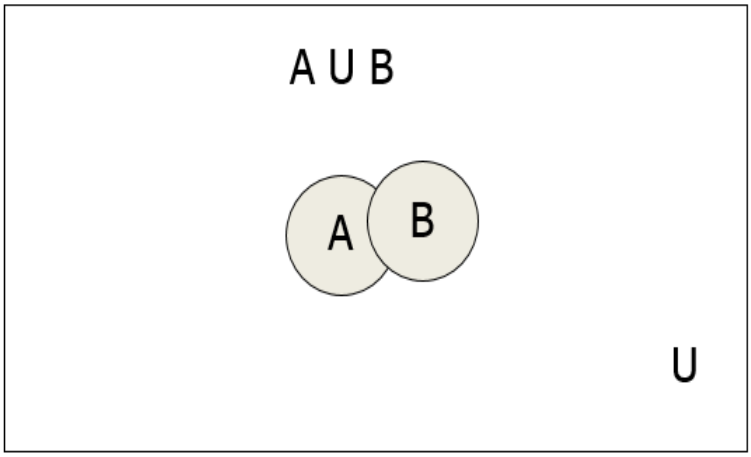

Figure 1.5: Venn diagram of the union of two sets.

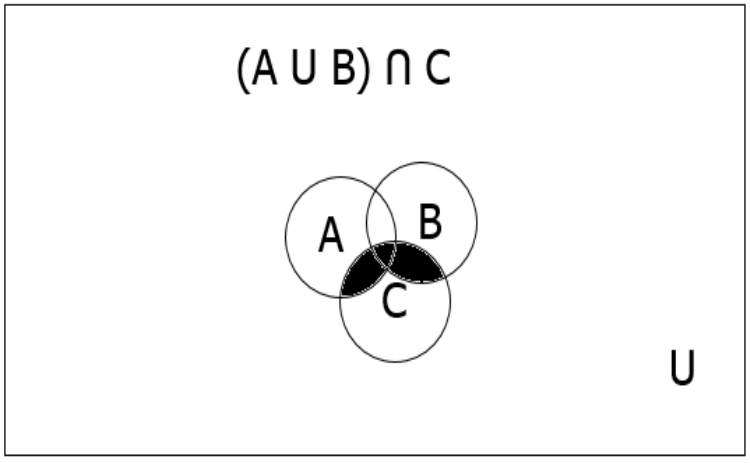

Figure 1.6: Venn diagram of a complicated intersection.

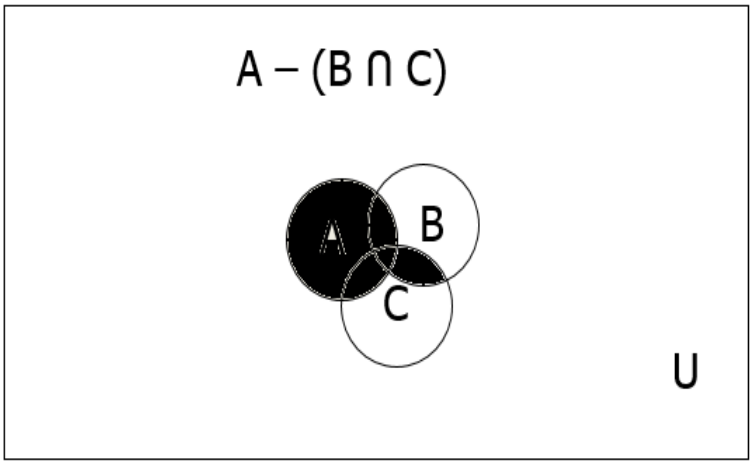

Figure 1.7: Venn diagram of a complicated negation.

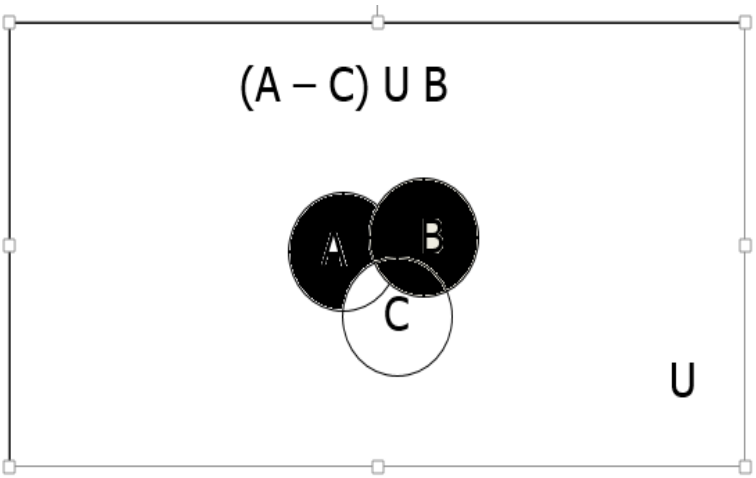

Figure 1.8: Venn diagram of negation and union.

This section covers Venn diagrams and De'Morgan's laws. Let $A \subset U \wedge B \subset U$. Then, the following are *De'Morgan's Laws:*

1. $\overline{A \cup B} = \overline{A} \cap \overline{B}$.
2. $\overline{A \cap B} = \overline{A} \cup \overline{B}$.

Venn diagrams are used to represent set operations. See Figures 1.3 thru 1.8.

Do problems 1a, b and 10 on page 92.

1.2.9 Homework and Answers

Page 92 of the textbook.

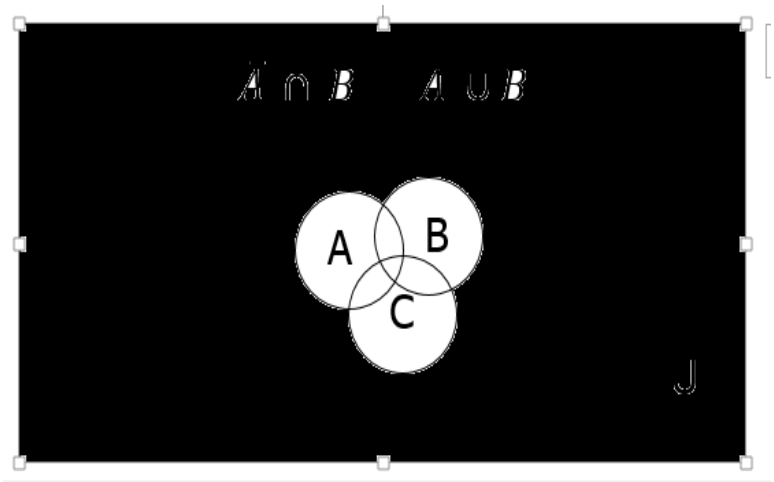

Figure 1.9: Venn diagram of Problem 1a $\overline{A} \cap \overline{B} = \overline{A \cup B}$.

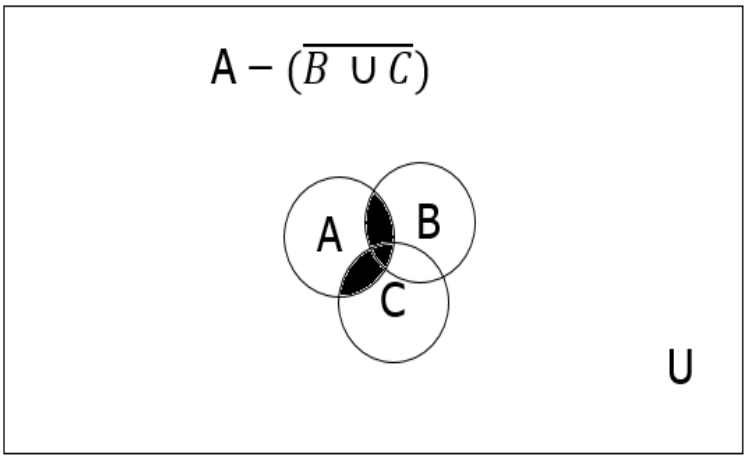

Figure 1.10: Venn diagram of Problem 1a $A - (\overline{B \cup C})$.

1. Construct Venn diagrams for each of the following.

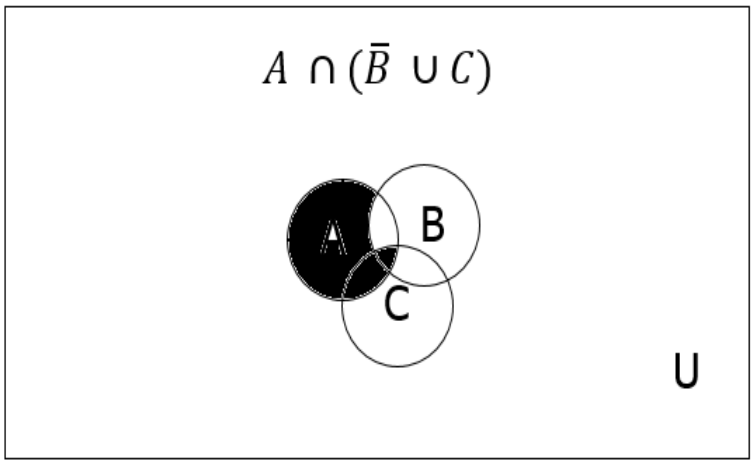

Figure 1.11: Venn diagram of Problem 1a $A \cap (\overline{B} \cup C)$.

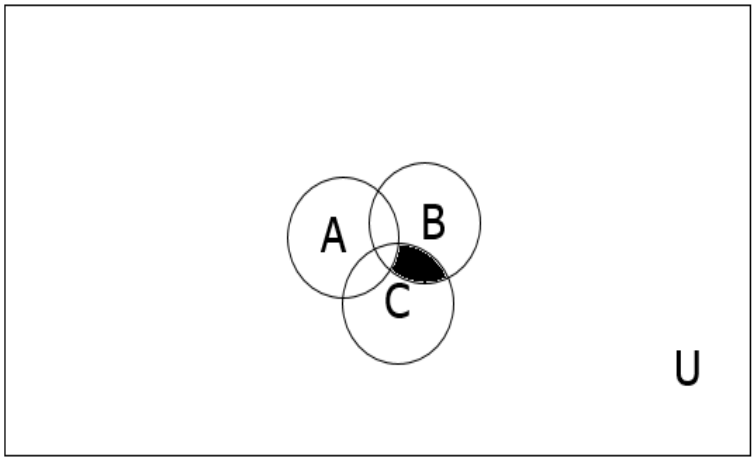

Figure 1.12: Venn diagram of Problem 1b (i).

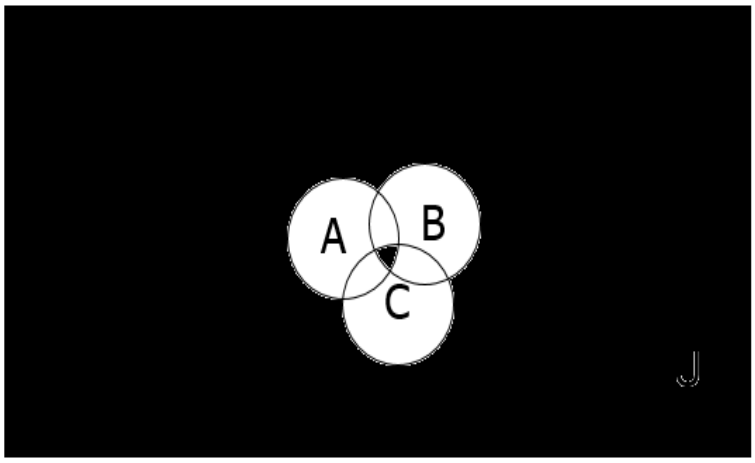

Figure 1.13: Venn diagram of Problem 1b (ii).

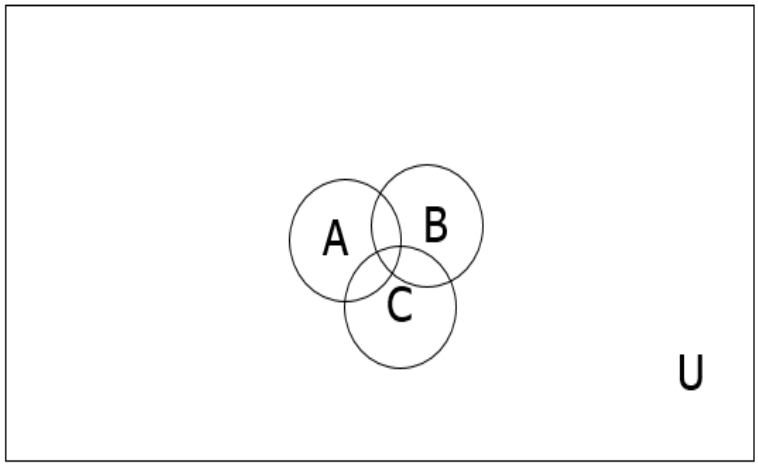

Figure 1.14: Venn diagram of Problem 1b (iii).

a. $A \cup B$. Solution: See Figure 1.5. $\overline{A} \cap \overline{B} = \overline{A \cup B}$. See Figure 1.9. $A - (\overline{B \cup C})$. See Figure 1.10. $A \cap (\overline{B} \cup C)$. See Figure 1.11.

b. Give a formula which denotes the shaded portion of each of the following Venn diagrams. Solution: See Figure 1.12. (i) $(B \cap C) - A$. See Figure 1.13. (ii) $(\overline{A \cup B \cup C}) \cup (A \cap B \cap C)$. See Figure 1.14. (iii) $\emptyset$.

10. Prove the following identities.

a. $A \cup (A \cap B) = A$. Solution:
$x \in A \cup (B \cap A)$
$x \in (A \vee (B \wedge A))$
$x \in (A \vee B) \wedge x \in (A \vee A)$
$x \in (A \vee B) \wedge x \in A$
$x \in (A \wedge A) \vee x \in (A \wedge B)$
$x \in A \vee x \in (A \wedge B)$.

b. $A \cap (A \cup B) = A$. Solution:
$x \in A \cap (A \cup B)$
$x \in A \wedge x \in (A \vee B)$
$(x \in A \wedge x \in A) \vee (x \in A \wedge x \in B)$
$x \in A \vee x \in (A \wedge B)$
$x \in A$
$\therefore A$.

c. $A - B = A \cap \overline{B}$. Solution:
$x \in A - B$
$x \in A \wedge x \notin B$
$x \in A \cap \overline{B}$
$A \cap \overline{B}$.

d. $A \cup (\overline{A} \cap B) = A \cup B$. Solution:
$x \in A \cup (\overline{A} \cap B)$
$x \in A \vee (x \notin A \wedge x \in B)$

$(x \in A \vee x \notin A) \wedge (x \in A \vee x \in B)$
$1 \wedge x \in (A \vee B)$
$x \in (A \cup B)$
$(A \cup B)$.

e. $A \cap (\overline{A} \cup B) = A \cap B$. Solution:
$x \in A \cap (\overline{A} \cup B)$
$x \in A \cap x \in (\overline{A} \cup B)$
$x \in A \cap (x \notin A \cup x \in B)$
$x \in A \wedge (x \notin A \vee x \in B)$
$(x \in A \wedge x \notin A) \vee (x \in A \wedge x \in B)$
$0 \vee (x \in A \wedge x \in B)$
$x \in (A \wedge B)$
$x \in (A \cap B)$
$A \cap B$.

1.2.10 Generalized Unions and Intersections

Test on Section 1.5. Direct proof, indirect proof, vacuous proof, trivial proof, contradition proof, constructive proof. Section 2.1, 2.3 and 2.4. Set operations, subsets and Venn diagrams. Prove page 88 but not f and g.

If C is a non-empty collection of subsets of U, then

1. The union of the elements of C is $\cup_{s \in C} S = \{x | \exists s[s \in C \wedge x \in S\}$.

   Example: $C = \{\{1,2,3\},\{3,4\},\{2,5,8\}\}$. $\cup_{s \in C} S = \{1,2,3,4,5,8\}$.

2. The intersection of the elements of C is $\cap_{s \in C} S = \{x | \forall s \ \ s \in C \Rightarrow x \in S\}$.
   Also, $\cap_{s \in C} S = \emptyset$.

The *power set* of a set A is denoted by $P(A)$ and is the set of all subsets of A.

Example: $A = \{a, b\}$. $P(A) = \{\emptyset, \{a,b\}, \{a\}, \{b\}\}$.

Do problems 11, 14, and 15 on page 94 in the textbook.

1.2.11 Induction

Inductively defined sets are all infinite. An *inductive definition* has 3 components:

1. *The basis clause* — it establishes that certain elements are in the set.

2. *The inductive clause* — this describes how elements of the set can be combined to form new elements of the set.

3. *The extremal clause* — this asserts that the only elements in the set are those which can be generated by applying clauses (1) and (2) a finite number of times.

Example: Define the set of even, non-negative integers $E = \{0, 2, 4, 6, ...\}$.

1. Basis — $0 \in E$.

2. Inductive — If $n \in E$, then $n + 2 \in E$.

3. Extremal — All elements in E can be generated by a finite number of applications of caluses (1) and (2).

Prove that $8 \in E$. Solution:
$0 \in E$. Basis
$0 \in E \Rightarrow 0 + 2 \in E$. Inductive
$2 \in E \Rightarrow 2 + 2 \in E$. Inductive
$4 \in E \Rightarrow 4 + 2 \in E$. Inductive
$6 \in E \Rightarrow 6 + 2 \in E$. Inductive

An *alphabet* denoted by Σ is a finite non-empty set of symbols or characters. A *word* or *a string* over Σ is a string of finite number of symbols from Σ.

Example: If $x = a_1 a_2 \cdots a_n$ is a string over sigma (Σ), the length of x is n.

The *null string* or *empty string* is denoted by Λ and is the string of length 0. If x and y are strings over Σ and $x = a_1 a_2 \cdots a_n$ and $y = b_1 b_2 \cdots b_m$, then x *concatenated* with y is denoted by xy is the string $xy = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m$. Note that concatenation is not commutative. If Σ is an alphabet, then Σ^+ denotes a set of all non-empty strings over sigma (Σ).

1. Basis — if $a \in \Sigma$, then $a \in \Sigma^+$.
2. Inductive — if $x \in \Sigma^+$, and $a \in \Sigma$, then $ax \in \Sigma^+$.
3. Extremal — Σ^+ contains only those strings which can be generated by a finite number of applications of (1) and (2).

Example: $\Sigma = \{a, b\}$. Prove $babba \in \Sigma^+$. Go right-to-left.
$a \in \Sigma^+$. Basis
$ba \in \Sigma^+$. Inductive
$bba \in \Sigma^+$. Inductive
$abba \in \Sigma^+$. Inductive
$babba \in \Sigma^+$. Inductive

The *transitive closure* Σ^* of Σ is the set of all finite length strings over Σ. $\Sigma^* = \Sigma^+ \cup \{\Lambda\}$. The inductive definition of Σ^* is as follow:

1. Basis — $\Lambda \in \Sigma^*$.
2. Inductive — If $x \in \Sigma^*$ and $a \in \Sigma$, then $ax \in \Sigma^*$.
3. Extremal — The same.

For any string x over Σ, $x\Lambda = \Lambda x = x$.

Example: $\Sigma = \{a, b\}$. $\Sigma^* = \{\Lambda, a\Lambda, b\Lambda, ...\}$.

1.2.12 Inductive Definition of Sets

This section covers the bottom of page 97 of the textbook and page 98, part A.

Let S equal to well formed arithmetic expressions. $S = \{0, 1, 2, ..., 9, 10, 11, 12, 13, ..., (+0), (2 + 9), \left(\frac{31}{42}\right), \left((2 + 0 \times \left(\frac{31}{42}\right)\right), ...\}$. Part B: $V = \{P, Q, R, ...\}$.. Basis: $\{0, 1, P, Q, R, (P \vee Q), (\neg(P \vee Q))$.

Inductive proofs: These are proofs of assertions of the form $\forall x P(x)$. The universe is inductively defined. The proofs have 2 parts:

1. Basis — show that $P(x)$ is true for all elements in the basis part of the definition.

2. Induction — show that every element obtained using the indutive clause of the difinition satisfies $P(x)$ of all elements used in its construction also satisfies $P(x)$.

Example: Let U = natural numbers = $\{0, 1, 2, ...\}$. The natural numbers N are defined as follow:

1. $0 \in N$.

2. If $n \in N$, then $n + 1 \in N$.

3. Same.

To use this definition to prove $\forall x P(x)$, show that

1. $P(0)$ is true

2. Show $\forall n[P(n) \Rightarrow P(n + 1)]$.

This is called the *first principle of mathematical induction.*

Example: $\forall n \in N$. $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$. Proof:

1. If $n = 0$, $\sum_{i=0}^{0} = \frac{0(1)}{2} = 0 = 0$.

2. Assume $P(n)$ is true. Show that $P(n + 1)$ is also true.

$$\sum_{i=0}^{n+1} i = (n + 1) + \sum_{i=0}^{n} i =$$

$$\frac{2(n + 1)}{2} + \frac{n(n + 1)}{2} =$$

$$\frac{2(n + 1) + n(n + 1)}{2} =$$

$$\frac{(n + 1)(n + 2)}{2}$$

$\therefore \forall n P(n)$.

For the first principle of mathematical induction over natural numbers:

(a) Show that $P(0)$ is true.

(b) Assume $P(n)$ is true, then show $P(n + 1)$ is true.

Applied to binary trees, this gives the following result. If a binary tree has n nodes, then there are $n + 1$ nil pointers. Proof:

(a) If $n = 1$, then there are 2 nil pointers.

(b) Assume if there n nodes, then there are $n + 1$ nil pointers. Show if there are $n + 1$ nodes then there are $n + 2$ nil pointers. Add 1 node to a tree with n nodes. The original tree had $n + 1$ nil pointers. By adding 1 node, one nil pointer is lost, but 2 nil pointers are gained. So, the total gain is 1 nil pointer. $\therefore$ There are now $n + 2$ nil pointers.

The *second principle of induction:* Assume $P(k)$ is true for all $k < n$. Show that $P(n)$ is also true.

```
procedure inorder(p: pointer);
begin
if p <> nil then
begin
inorder(p↑ .left);
process(p);
inorder(p↑ .right);
end;
end;
```

Prove that the procedure INORDER correctly visits the nodes in a tree of size n. Proof: Assume that INORDER works correctly on all trees with fewer than n nodes. Each node has at most $n-1$ nodes.

1.2.13 Set Operations on Σ^*

Set operations on Σ^*, where Σ^* is equal to the set of all finite length strings over Σ (chosen characters). Note that $\Lambda \in \Sigma^*$. Let $x \in \Sigma^*$ and $n \in N$, then

(a) Basis — $x^0 = \Lambda$.

(b) Induction — $x^{n+1} = x^n x$.

Example: $\Sigma = \{a, b, c\}$. Choose $x = abc \in \Sigma^*$.
$x^0 = (abc)^0 = \Lambda$
$x^1 = x^0 \cdot x = \Lambda \cdot x = abc$
$x^2 = x^1 \cdot x = abc \cdot abc = abcabc.$

Example: $\{(a, b)^n | n \geq 0\} = \{\Lambda, ab, abab, ababab, ...\}$.

Example: $\{(a^n b^n | n \geq 0\} = \{\Lambda, ab, aabb, aaabbb, ..., \}$.

Let Σ be a finite alphabet. Then a language over Σ is any subset of Σ^*.

Example: $\Sigma = \{a, b, c\}$. Let $A = \{ab, acc, b, ca\}$. A is a *language* over Σ.

Σ is equal to the set of ASCII characters. Σ^* is equal to all finite strings of ASCII characters. Let A equal to the set of all legal Pascal commands or programs. If A and B are languages over Σ, then the *set product* of A with B is the language consisting of all strings formed by concatinating an element of A with an element of B.

Example: $\Sigma = \{a, b, c\}$. $A = \{a, bc\}$. $B = \{\Lambda, b, ac\}$. Then, $AB = \{a, ab, aac, bc, bcb, bcac\}$. Also, $AB \neq BA$.

Quiz on Monday: Induction and sets. Don't have to know Section 2.7.1 on 113. Know Section 2.7.3 on page 115. Have to know 1 induction proof like the one on page 107 #4 and 6. Given an induction definition — work with it — prove an element is in the set. Know Section 2.5 and 2.7

1.2.14 Homework and Answers

Page 106 - 107, 1, 3, 4, 6a, b, c in the textbook.

1. Give inductive definitions for the following sets.

a. The set of unsigned integers in decimal representation. The defined set should include 4, 167, 0012, etc. Solution: Unsigned integers equals to S. Basis: let $D = \{0, 1, 2, ..., 9\}$. If $d \in D$, then $d \in S$. Induction: If $x \in S$, and $d \in D$, then $xd \in S$. External: 271 is an unsigned integer. So, Step:

(a) $2 \in S$.
(b) $2 \in S$, $7 \in D \Rightarrow 27 \in S$.
(c) $27 \in S$, $1 \in D \Rightarrow 271 \in S$.

b. The set of real numbers with terminating fractional parts in decimal representation. The defined set should include 6.1, 712., 01.2100, 0.190, etc. Solution: Real number with terminating fractional parts equal to S. Basis: let $D = \{0, 1, 2, ..., 9\}$. If $d \in D$, then $d\cdot \in S$ and $\cdot d \in S$. Induction: If $x \in S$ and $d \in D$, then $xd \in S$ and $dx \in S$. Extremal: consider 21.68. So,
Step:

(a) $1. \in S$.
(b) $1. \in S \wedge 2 \in D \Rightarrow 21. \in S$.
(c) $21. \in S \wedge 6 \in D \Rightarrow 21.6 \in S$.
(d) $21.6 \in S \wedge 8 \in D \Rightarrow 21.68 \in S$.

3. Give an inductive definition of $n!$ and use it to prove the identity

$$n! = \prod_{i=1}^{n} i.$$

Solution: $n! = n(n-1)(n-2)\cdots 3 \cdot 2 \cdot 1$. Basis: $1! = 1$. Induction: show $\prod_{i=1}^{n+1} i = (n+1)!$

$$\prod_{i=1}^{n+1} i = (n+1)\prod_{i=1}^{n} i =$$
$$(n+1)n! =$$
$$(n+1)!$$

4. Prove by induction that $(1 + 2 + 3 + \cdots + n)^2 = 1^3 + 2^3 + 3^3 + \cdots + n^3$ for all $n \in I+$. Solution: Proof basis case. $n = 1$, $P(1) = 1^2 = 1^3 = 1$. Induction: Assume $P(n)$ is true. Show $P(n+1)$ is true.

$$P(n+1) =$$
$$(1 + 2 + \cdots + n + (n+1))^2 =$$
$$(1^3 + 2^3 + 3^3 + \cdots + (n+1)^3) =$$
$$(1 + 2 + \cdots + n + 1)^2 =$$
$$((1 + 2 + \cdots + n) + (n+1))^2 =$$
$$(1 + 2 + 3 + \cdots + n)^2 + 2(1 + 2 + \cdots + n)(n+1) + (n+1)^2 =$$
$$1^3 + 2^3 + \cdots + n^3 + \frac{2n(n+1)(n+1)}{2} + (n+1)^2 =$$
$$1^3 + 2^3 + \cdots + n^3 + n(n+1)^2 + (n+1)^2 =$$
$$1^3 + 2^3 + \cdots + n^3 + (n+1)^2(n+1) =$$
$$1^3 + 2^3 + \cdots + n^3 + (n+1)^3.$$

6. Prove each of the following relationships for all $n \in N$.

a. $\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$. Basis: $n = 0$. $\sum_{i=0}^{0} i^2 = \frac{0(0+1)(0+1)}{6} = 0 = 0$. Induction: Assume $\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$. Show that $\sum_{i=0}^{n+1} i^2 = \frac{(n+1)(n+2)(2n+3)}{6}$.

$$\sum_{i=0}^{n+1} i^2 = (n+1)^2 + \sum_{i=0}^{n} i^2 =$$
$$(n+1)^2 + \frac{n(n+1)(2n+1)}{6} =$$
$$\frac{6(n+1)^2}{6} + \frac{n(n+1)(2n+1)}{6} =$$
$$\frac{6(n+1)^2 + n(n+1)(2n+1)}{6} =$$
$$\frac{(n+1)[6(n+1) + n(2n+1)]}{6} =$$
$$\frac{(n+1)[6n+6+2n^2+n]}{6} =$$
$$\frac{(n+1)[2n^2+7n+6]}{6} =$$
$$\frac{(n+1)(n+2)(2n+3)}{6}.$$

b. $\sum_{i=0}^{n}(2i+1) = (n+1)^2 = P(n)$. Solution: For the basis case, show that $P(0) = 0$. $\sum_{i=0}^{0}(2i+1) = (0+1)^2$, $1 = 1$. For the induction case, show $P(n+1)$ is also true.

$$P(n+1) = \sum_{i=0}^{n+1}(2i+1) = (n+2)^2$$
$$\sum_{i=0}^{n+1}(2i+1) = (2(n+1)+1) + \sum_{i=0}^{n} 2i + 1 =$$
$$(2n+3) + (n+1)^2$$
$$2n + 3 + n^2 + 2n + 1 =$$
$$n^2 + 4n + 4 =$$
$$(n+2)^2.$$

c. $\sum_{i=0}^{n} i(i!) = (n+1)! - 1$. Solution: Show $\sum_{i=0}^{n} i(i!) = (n+1)! - 1$. Basis: if $n = 0$, $\sum_{i=0}^{0} i(i!) = (0+1)! - 1 = 0(0!) - 1 = 0 = 0$. Induction: Assume that $\sum_{i=0}^{n} i(i!) = (n+1)! - 1$. Show $\sum_{i=0}^{n+1} i(i!) = (n+2)! - 1$.

$$\sum_{i=0}^{n+1} i(i!) =$$
$$(n+1)(n+1)! + \sum_{i=0}^{n} i(i!) =$$
$$(n+1)(n+1)! + (n+1)! - 1 =$$
$$(n+1)![(n+1)+1] - 1 =$$
$$(n+1)![(n+2)] - 1 =$$
$$(n+2)! - 1.$$

1.3 Binary Relations

An ordered $n-$tuple is a sequence of n objects denoted by $< a_1, a_2, ..., a_n >$ where a_i represents the i^{th} component of the n tuple. If $n = 2$, we have an *ordered pair.* If $n = 3$, we have an *ordered triple.* The *Cartesian product* of sets $A_1, A_2, ... A_n$ is denoted by $A_1 \times A_2 \times A_3 \times \cdots \times A_n$ and is given by $\{< a_1, a_2, ..., a_n > | a_i \in A_i\}$.

Example: $A = \{2, 4\}$. $B = \{a, b, c\}$. $A \times B = \{< 2, a >, < 2, b >, < 2, c >, < 4, a >, < 4, b >, < 4, c >\}$. $B \times A = \{< a, 2 >, < b, 2 >, < c, 2 >, < a, 4 >, < b, 4 >, < c, 4 >\}$. $A \times B \neq B \times A$. Let $C = \{6\}$. $ABC = A \times B \times C = \{< 2, a, 6 >, < 2, b, 6 >, < 2, c, 6 >, < 4, a, 6 >, < 4, b, 6 >, < 4, c, 6 >\}$. $A^2 = A \times A = \{< 2, 2 >, < 2, 4 >, < 4, 2 >, < 4, 4 >\}$. $A^n = A \times A \times A \times \cdots \times A$.

$\Re$ equals to the real numbers. $\Re^2 = R \times R = \{< x, y > | x, y \in \Re\}$. A relation, R, is an $n-$ariry relation on $\prod_{i=1}^{n} A_i$ if R is a subset of $\prod_{i=1}^{n} A_i$. If $R = \emptyset$, then we say R is the *empty relation.* If $R = \prod_{i=1}^{n} A_i$, then we say R is the *universal relation.* If $n = 1$, then R is a *unary relation.* If $n = 2$, then R is a *binary relation.* If R is a binary relation on A, then $R \subset A \times A$. Let P be a predicate. Define P as $P < a_1, a_2, a_3, ..., a_n >$ is true iff $< a_1, a_2, ..., a_n > \in R$.

Example: Let A = natural numbers $= \{0, 1, 2, ...\}$. Let $P(x, y)$ mean $x < y$. P corresponds to a binary relation on A. $R = \{< x, y > | x < y\}$.

Some notation: $< a, b > \in R$ is donoted by $a \ R \ b$. Read page 124 and the top of page 125 in the textbook. There are 4 examples. Define $<$ (less-than relation). Basis: $0 < 1$. Induction: if $x < y$, then $x < y + 1$. $x + 1 < y + 1$.

R is a binary relation on set A if $R \subset A \times A$. $\{< x, y > | x, y \in A\}$.

1.3.1 Graph Theory

A *directed graph (digraph)* is an ordered pair of the form $D =< A, R >$ where A is a set and R is a binary relation on A. $R \subset A \times A$. Elements of A are the *nodes* or *vertices* of the graph. Elements of R are the *edges* or *arcs* or *lines* of the graph. We will assume that A is finite. If $< x, y > \in R$, then we write $x \ R \ y$. This will mean that there is an arrow from x to y in the graph.

Example: $A = \{a, b, c\}$. $R = \{< a, b >, < b, c >, < c, c >\}$. $G :< A, R >$ is a graph.

To represent a graph in a computer:

1. Incidence matrix

	a	b	c
a	0	1	0
b	0	0	1
c	0	0	1

   This representation takes alot of space.

2. Adjacency list

An *edge* orginates at a and *terminates* at b. $A = \{a, b, c\}$. $D =< A, R >$. $R \subset A \times A$. $R = \{< a, b >, < a, c >, < b, a >, < b, c >, < c, c >\}$. Let $D =< A, R >$ be a directed graph with $a, b \in A$. An *undirected path* P from a to b is a finite sequence of verticies such that $P =< c_0, c_1, c_2, ..., c_n >$ such that 3 things are true:

1. $c_0 = a$.

2. $c_n = b$.

3. Either $c_i \ R \ c_{i+1}$ or $c_{i+1} \ R \ c_i, \forall 0 \leq i \leq n$. The latter is not in directed graphs. The path from b to c for undirected graphs is $< b, c >, < b, a, c >, < b, a, b, c, c >$. The path from c to b is undirected.

If P is a directed path from a to b then:

1. Vertex a is the initial vertex. Vertex b is the *terminal vertex.*
2. The length of the path is n edges.
3. If all the verticies of P are distinct except possibly c_0 and c_n, then P is a *simple* or *cordless path.*
4. If $c_0 = c_n$, then P is a cycle.

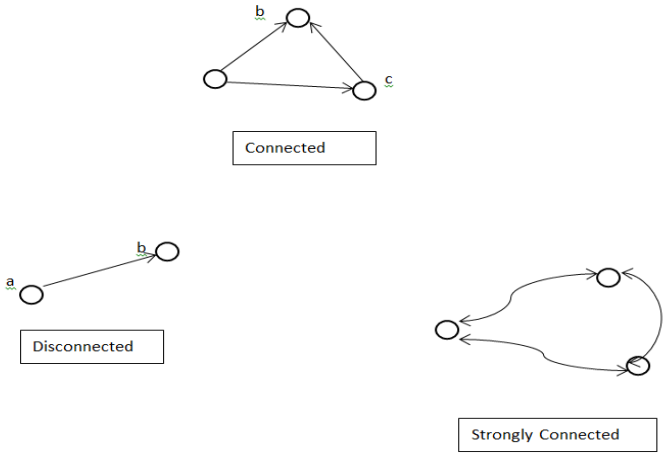

Figure 1.15: Diagram of a disconnected, strongly connected, and connected graphs.

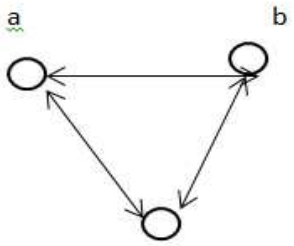

Figure 1.16: Diagram of a complete graph.

Digraph $D =< A, R >$ is *strongly connected* if for all verticies, $\forall a, b \in A$, there is a directed path from a to b and from b to a. D is *connected* if $\forall a, b \in A$ there is an undirected path from a to b. D is *disconnected* if there exist verticies $a, b \in A$ such that there is no undirected path between a and b.

Example: See Figure 1.15.

D is *complete over* A if $R = A \times A$. See Figure 1.16.

Let R be a binary relation on set A. $R \subset A \times A$. Then,

1. R is *reflexive* if $x \ R \ x \ \ \forall x \in A$. Loops.

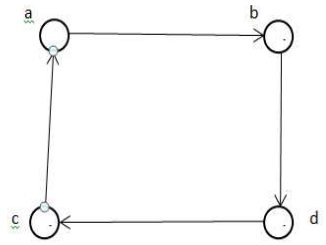

Figure 1.17: Diagram A for problem 2 on page 130 in the textbook.

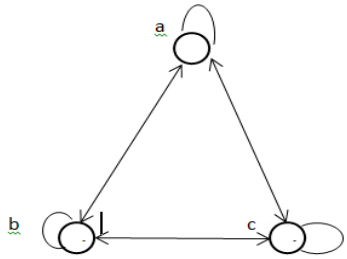

Figure 1.18: Diagram B for problem 2 on page 130 in the textbook.

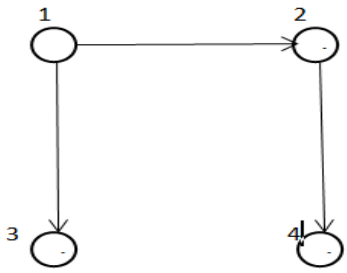

Figure 1.19: Diagram of the solution to homework question 3a on page 130 in the textbook.

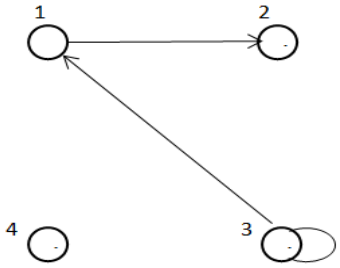

Figure 1.20: Diagram of the solution to homework question 3b on page 130 in the textbook.

2. R is *irreflexive* if $x \not R\ x\ \forall x \in A$. No loops.
3. R is *symmetric* if $x\ R\ y \Rightarrow y\ R\ x \forall x, y \in A$. 2 edges.
4. R is *antisymmetric* if $(x\ R\ y \wedge y\ R\ x) \Rightarrow (x = y) \forall x, y \in A$. No double edges.
5. R is *transitive* if $(x\ R\ y \wedge y\ R\ z) \Rightarrow x\ R\ z \forall x, y, z \in A$.

Do problems on page 130, #1, 2a, b, c, 3, 4, 5, 6.

1.3.2 Homework and Answers

Problems on page 130 in the textbook. #1, 2a, b, c, 3, 4, 5, 6.

1. Let $A = \{0, 1, 2, 3, 4\}$. For each of the predicates given below, specify the set of n-tuples in the n-ary relation over A which corresponds to the predicate. For parts (d) - (f), draw the diagram which represents the relation.

a. $P(x) \Leftrightarrow x \leq 1$. Solution: $R = \{< 0 >, < 1 >\}$.

b. $P(x) \Leftrightarrow 3 > 2$. Solution: $\{< 0 >, < 1 >, < 2 >, < 3 >, < 4 >\}$.

c. $P(x) \Leftrightarrow 2 > 3$. Solution: Always false $\emptyset$.

e. $P(x, y) \Leftrightarrow \exists k[x = ky \wedge k < 2]$. Solution: $K = 0, 1$. $R = \{< 0, 0 >, < 0, 2 >, < 0, 3 >, < 0, 4 >, < 1, 1 >, < 2, 2 >, < 3, 3 >, < 4, 4 >\}$.

2. For the following digraphs A and B in Figures 1.17 and 1.18.

a. Find all simple paths from node a to node c. Give the path lengths. Solution: Graph A. $< a, b, d, c >$. $n = 3$. Graph B $\{< a, c >\}$, $\{< a, b >, < b, c >\}$.

b. Find the indegree and outdegree of each node. Solution: Graph A. Indegree a 1, b 1, and c 1. Outdegree a 1, b 1, and c 1. Graph B. Indegree a 3, b 3, and c 3. Outdegree a 3, b 3, and c 3.

c. Find all simple cycles with initial and terminal node a. Solution: Graph $A : < a, bd, c, a >, < a >$. Graph $B : \{< a, b, c, a >\}$ $\{< a, c, b, a >\}$.

3. For each of the following, sketch a digraph of the given binary relation on A. State whether the digraph is disconnected, connected, or strongly connected and state how many components the digraph has.

a. $[< 1, 2 >, < 1, 3 >, < 2, 4 >]$ where $A = \{1, 2, 3, 4\}$. Solution: Connected. See Figure 1.19.

b. $[< 1, 2 >, < 3, 1 >, < 3, 3 >]$ where $A = \{1, 2, 3, 4\}$. Solution: Disconnected. See Figure 1.20.

c. $[< x, y > | 0 \leq x < y \leq 3]$ where $A = \{0, 1, 2, 3, 4\}$. Solution: Disconnected.

e. $[< x, y > | 0 \leq x - y < 3]$ where $A = \{0, 1, 2, 3, 4\}$. Solution: $R = \{< 0, 0 >, < 1, 0 >, < 1, 1 >, < 2, 0 >, < 2, 1 >, < 2, 3 >, < 3, 1 >, < 3, 2 >, < 3, 3 >, < 4, 2 >, < 4, 3 >, < 4, 4 >\}$.

4. Construct the incidence matrix for the following binary relation on $[0, 1, 2, 3, 4, 5, 6] : \{< x, y > | x < y \vee x$ is prime $\}$. Solution: Connected.

1	2	3	4	5	6	7	8
0	0	1	1	1	1	1	1
1	0	0	1	1	1	1	1
2	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1
4	0	0	0	0	0	1	1
5	1	1	1	1	1	1	1
4	0	0	0	0	0	0	0

5. For each of the following, give an inductive definition for the relation R on N. In each case, use your definition to show $x \in R$.

a. $R = \{< a, b > | a \geq b\}; x =< 3, 1 >$. Solution: Basis: $0 \geq 0$ or $< 0, 0 > \in R$ or $0\ R\ 0$. Induction: if $x \geq y$, then $x + 1 \geq y$ and $x + 1 \geq y + 1$. For example, prove that $5 \geq 3$.
$0 \geq 0$ basis
$1 \geq 0$
$2 \geq 0$
$3 \geq 1$
$4 \geq 2$
$5 \geq 3$.

b. $R = \{< a, b > | a = 2b\}; x =< 6, 3 >$. Solution: Basis: $< 0, 0 > \in R$. Induction: if $< x, y > \in R$, then $< x + 2, y + 1 > \in R$.

c. $R = \{< a, b, c > | a + b = c\}; x =< 1, 1, 2 >$. Solution: Basis: $< 0, 0, 0 > \in R$. Induction: If $< x, y, z > \in R$ then $< x + 1, y, z + 1 > \in R$. If $< x, y, z > \in R$ then $< x, y + 1, z + 1 > \in R$. Show that $< 2, 3, 5 > \in R$.
$< 0, 0, 0 > \in R$
$< 1, 0, 1 > \in R$
$< 2, 0, 2 > \in R$
$< 2, 1, 3 > \in R$
$< 2, 2, 4 > \in R$
$< 2, 3, 5 > \in R$.

6. Let $A = \{1, 2, 3\}$.

a. List the unary relation on A. Solution: If R is a unary relation on A, then $R \subset A$. $R = \emptyset$. $R = \{< 1 >, < 2 >, < 3 >\}$. $R = \{< 1 >\}, \{< 2 >\}, \{< 3 >\}, \{< 1, 2 >\}, \{< 1, 3 >\}, \{< 2, 3 >\}$.

b. How many binary relations are there on A? Solution: $R \subset A \times A$ is a binary relation. $A \times A$ has 9 elements. $\therefore 2^9$ subsets $= 512$.

1.3.3 Homework and Answers

Figure 1.21: Diagram to homework question 1a on page 147 in the textbook.

Figure 1.22: Diagram to homework question 1b on page 147 in the textbook.

Page 147 and 148 in the textbook.

1. List the properties defined in Definition 3.3.1 which hold for the relations represented by the following graphs.

a. See Figure 1.21. Solution: Not reflexive. Not irreflexive. Not symmetric. Antisymmetric. Transitive.

b. See Figure 1.22. Solution: Relfexive. Not irreflexive. Symmetric. Not antisymmetric. Transitive.

3. Consider the set of integers I. Fill in the following table with Yes and No according to whether the relation possesses the property. The notation $\emptyset$ denotes the empty set, $I \times I$ is the universal relation, and D denotes "divides with an integer quotient" (e.g. $4D8$ but $4\not D7$). Solution: $D : R = \{< x, y > | x$ divides $y\}$. For example, $< 4, 8 > \in R$.

$I \times I$	$\leq$	D
Reflective	Reflexive	Not reflexive
Not Irreflective	Not Irreflective	Not Irreflective
Symmetric	Not symmetric	No
Not antisymmetric	Antisymmetric	No
Transitive	Transitive	Yes

1.3.4 Composition of Relations

Let R_1 be a relation from A to B. $R_1 \subset A \times B$. Let R_2 be a relation from B to C. $R_2 \subset B \times C$. Then, the *composite relation* from A to C is denoted by $R_1 \cdot R_2$ or R_1R_2 and is given by $R_1R_2 = \{< a, c > | a \in A \wedge c \in C \wedge \exists b[b \in B \wedge < a, b > \in R_1 \wedge < b, c > \in R_2]\}$.

Example: $A = \{0, 1, 2\}$. $B = \{a, b, c\}$. $C = \{y, z\}$. $R_1 = \{< 0, a >, < 1, b >, < 1, c >, < 2, b >\}$. $R_2 = \{< a, 2 >, < b, y >, < b, z >, < c, y >\}$. $R_1R_2 = \{< 0, z >, < 1, y >, < 1, z >, < 2, y >, < 2, z >\}$. R_2R_1 is impossible.

If $R_1 \subset A \times B \wedge R_1 \subset B \times C$ then $R_1R_2 \subset A \times C$. $R_1R_2 = \{< a, c > | a \in A, c \in C, \exists b[b \in B \wedge < a, b > \in R_1 \wedge < b, c > \in R_2]\}$. Let R be a binary relation on set A. Then, the n^{th} *power of* R is denoted by R^n, is defined by

1. $R^0 = \{< x, x > | x \in A\}$ i.e. R^0 means "=."
2. $R^{n+1} = R^n \cdot R$.

Example: $A = \{a, b, c\}$. $R = \{< a, b >, < a, c >, < b, b >, < c, a >\}$. $R^0 = \{< a, a >, < b, b >, < c, c >\}$. $R^1 = R^0R = \{< a, b >, < a, c >, < b, b >, < c, a >\}$. $R^2 = R^1R = \{< a, b >, < a, a >, < c, b >, < c, c >, < b, b >\}$. $R^3 = R^2R = \{< a, b >, < a, c >, < b, b >, < c, b >, < c, a >\}$.

1.3.5 Closure Operations on Relations

Let R be a binary relation on set A. Then, the *reflexive closure* of R, $r(R)$, is the relation R' such that:

1. R' is reflexive.
2. R' is a super-set of R i.e. $R' \supset R$.
3. R' is the smallest relation which satisfies (1) and (2).

Example: $A = \{a, b, c, d\}$. $R = \{< a, b >, < b, b >, < b, d >, < c, c >, < d, a >\}$. R is not reflexive. $R' = r(R) = \{< a, b >, < b, b >, < b, d >, < c, c >, < d, a >, < a, a >, < d, d >\}$.

Let E be the binary relation of equality on any set A i.e. $E = \{< x, x > | x \in A\}$. Notice that $r(R) = R \cup E$ by a computer.

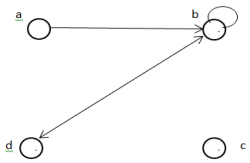

Figure 1.23: Diagram illustrating an example of adding arc to show the different relations.

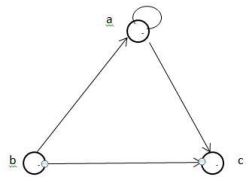

Figure 1.24: Diagram illustrating an example of reflexive, antisymmetric, transitive, and partial order.

Example: $A = \{a, b, c, d\}$. $R = \{< a, a >, < a, c >, < b, c >, < d, b >, < d, d >\}$. R is not symmetric. $A\ R\ c$ but $c\ \not{R}\ a$. $S(R) = \{< a, a >, < a, c >, < b, c >, < d, b >, < d, d >, < c, a >, < c, b >, b, d >\}$.

Let R be a binary relation on A. Then, R^c is called the *converse* of R and is given by $R^c = \{< y, x > | < x, y >\in R\}$. Result: $S(R) = R \cup R^c$ by a computer. Another result: R is symmetric iff $R = R^c$.

Example: $A = \{a, b, c\}$. $R = \{< a, b >, < b, a >, < a, c >\}$. R is not transitive. $t(R) = \{< a, b >< b, a >, < a, c >, < a, a >, < b, c >, < b, b >\}$. Note that $t(R) = R \cup R^n$.

Example: Consider Figure 1.23. $A = \{a, b, c, d\}$. Add $< a, a >, < c, c >, < d, d >$ to make it reflexive. Add $< b, c >$ to make it symmetric. Add $< a, d >, < d, a >$ to make it transitive.

Page 161 in the textbook, do problem 1.

Let R be a binary relation on set A. Then, R is a *partial order* if R is reflexive, antisymmetric, and transitive. The digraph $< A, R >$ is called a *partially ordered set* or *poset.* If R is a partial order on A then we write $a \leq b$, when $a\ R\ b$.

Example: " $\leq$ " on integers is a partial order.

Example: " $\subset$ " on the power $P(A)$ is a partial order. Let $T \in P(A)$. $T \subset T \Rightarrow$ reflexive. $T, S, R \in P(A)$. $T \subset S \wedge S \subset T \Rightarrow S = T \Rightarrow$ antisymmetric. $T \subset S \wedge S \subset R \Rightarrow T \subset R \Rightarrow$ transitive. See Figure 1.24

In a poset diagram, all loops are omitted; all edges implied by the transitive property are omitted; and all arrows are omitted. Let R be a binary relation on set A. R is a *quasi order* if R is transitive and irreflexive. Note it will also be antisymmetric. Does $(x\ R\ y \wedge y\ R\ x) \Rightarrow x = y$? If $x\ R\ y \wedge y\ R\ x$ then $x\ R\ x$ (transitive property) which is a contradiction by being irreflexive. Let $E = \{< x, x > |x \in A\}$. Results:

1. If R is a quasi-order then $R \cup E$ must be a partial order.
2. If R is a partial order then $R - E$ is quasi-order.

Partial order $(\leq)$ on set A is a *linear order* if $a \leq b$ or $b \leq a, \forall a, b \in A$. The digraph $< A, R >$ is called a *linear ordered set.*

Example: " $\leq$ " on I is a linear order.

Example: $A = \{a, b\}$. $P(A) = \{emptyset\}, \{a, b\}, \{a\}, \{b\}$. $R \sim' C'$. Not linear order.

If $< A, \leq >$ is a poset and if $B \subset A$, then:

1. $b \in B$ is a *greatest element* of B if $b' \leq b \forall b' \in B$.
2. $b \in B$ is a *least element* of B if $b \leq b' \forall b' \in B$.

Let R be a binary relation on A. Then R is a *well order* if R is a linear order and every non-empty subset of A has a least element. $< A, R >$ is called a *well ordered* set.

Example: " $\leq$ " on I but is well ordered on N. It is a linear order.

On page 175 in the textbook, do problems 1 and 3.

1.3.6 Homework and Answers

Page 153 of the textbook, Section 3.4

1. Let R_1 and R_2 be relations on a set $A = \{a, b, c, d\}$ where $R_1 = \{< a, a >, < a, b >, b, d >\}$ and $R_2 = \{< a, d >, < b, c >, < b, d >, < c, b >\}$. Find R_1, R_2, R_2R_1, R_1^1, and R_2^2. Solution: $R_1^2 = R_1^1 \cdot R_1 = \{< a, a >, < a, b >, < a, d >\}$.

9. Let R_1 and R_2 be arbitrary relations on a set A. Prove or disprove the following assertions.

a. If R_1 and R_2 are reflective, then R_1R_2 is reflective. Solution: True. b-e is False. R_1 is reflexive $\Rightarrow < x, x > \in R, \forall x \in A$. R_2 is reflexive $\Rightarrow < x, x > \in R_2 \forall x \in A$. $R_1R_2 < x, x > \forall x \in A$.

b. If R_1 and R_2 are irreflective, then R_1R_2 is irreflective. Solution: False. Find a counter example — one that is reflexive. $A = \{a, b, c\}$. $R_1 = \{< a, b >, < b, c >, < c, a >\}$. $R_2 = \{< b, a >, < c, b >, < b, c >\}$. $R_1 \cdot R_2 = \{< a, a >, < a, c >, < b, b >\}$. $\therefore R_1R_2$ is not irreflexive.

1.3.7 Homework and Answers

Page 161 of the textbook, Section 3.5

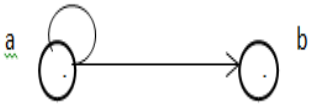

Figure 1.25: Diagram for problem 1b on page 161 in the textbook.

1b. Find the reflective, symmetric, and transitive closures of each of the graph in Figure 1.25. Solution: Reflexive $\{< b, b >\}$. Symmetric $\{< b, a >\}$. Already transitive.

1.3.8 Homework and Answers

Page 175, problems 1 and 3.

1. Fill in the following table describing the characteristics of the given ordered sets.

Figure 1.26: Diagram for problem 3a on page 175 in the textbook.

Figure 1.27: Diagram for problem 3b on page 175 in the textbook.

Figure 1.28: Diagram for problem 3c on page 175 in the textbook.

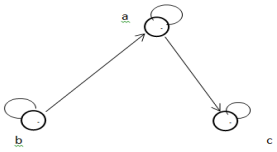

Figure 1.29: Diagram for problem 3d on page 175 in the textbook.

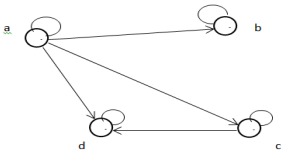

Figure 1.30: Diagram for problem 3e on page 175 in the textbook.

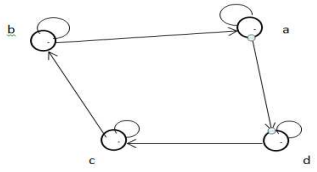

Figure 1.31: Diagram for problem 3f on page 175 in the textbook.

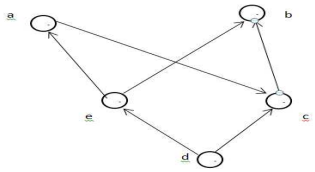

Figure 1.32: Diagram for problem 3g on page 175 in the textbook.

	Quasi Ordered	Partial Ordered	Linear Ordered	Well Ordered
$< N, <>$	Yes	No	No	No
$< N, \leq>$	No	Yes	Yes	Yes
$< I, \leq>$	No	Yes	Yes	No
$< R, \leq>$	No	Yes	Yes	No
$< P(N),$ proper containment $>$	Yes	No	No	No
$< P(N), \subset>$	No	Yes	No	No
$< P([a]), \subset>$	No	Yes	Yes	Yes
$< P([\emptyset]), \subset>$	No	Yes	Yes	Yes

3. State which of the following digraphs represent a quasi-order; a poset; a linearly ordered set; a well ordered set;

a. See Figure 1.26. Solution: Partial, not linear, not well.

b. See Figure 1.27. Solution: Partial, linear, well.

c. See Figure 1.28. Solution: No order. Symmetric.

d. See Figure 1.29. Solution: No order. Not transitive.

e. See Figure 1.30. Solution: Partial. Not linear $< b, d >, < b, c >$. Not well.

f. See Figure 1.31. Solution: Not partial — not transitive. Not linear, well quasi.

g. See Figure 1.32. Solution: No orders. Not reflexive or transitive.

The final exam will be Wednesday, December 9 from 1:00 to 2:00pm. It will cover Section 3.3 properties of binary relations; Section 3.4 Composition; also take powers of relations; Section 3.5 reflexive, symmetry, transitive closures of binary relation; Section 3.6 order relations. Office hours are Friday 10-12, Tues 10-12 and Weds 12-1.

1.3.9 Quiz 4

November 2, 1987

1. Using induction, prove the following $\sum_{i=1}^{n} i^3 = \left[\frac{n(n+1)}{2}\right]^2$. Solution: basis, let $n = 1$. Then $\sum_{i=1}^{1} i^3 = \left[\frac{1(1+1)}{2}\right]^2 = 1^3 = 1^2 = 1 = 1$. Induction: assume

$$\sum_{i=1}^{n} i^3 = \left[\frac{n(n+1)}{2}\right]^2 \text{ is true.}$$

Show

$$\sum_{i=1}^{n+1} i^3 = \left[\frac{(n+1)(n+2)}{2}\right]^2$$

is true.

$$\sum_{i=1}^{n+1} i^3 = \sum_{i=1}^{n} i^3 + (n+1)^3 =$$

$$\left[\frac{n(n+1)}{2}\right]^2 + (n+1)^3 =$$

$$\left[\frac{n(n+1)}{2}\right]\left[\frac{n(n+1)}{2}\right] + [(n+1)(n+1)(n+1)] =$$

$$(n+1)^2\left[\frac{n^2}{4} + n + 1\right] =$$

$$(n+1)^2\left[\frac{n^2}{4} + \frac{4n}{4} + \frac{4}{4}\right] =$$

$$(n+1)^2\left[\frac{n+2}{2}\right]^2 =$$

$$\left[\frac{(n+1)(n+2)}{2}\right]^2.$$

2. Define a set S inductively as follows:

   (a) $1 \in S$.

   (b) If $x \in S$, then $x1 \in S$.
   If $x \in S$, then $0x \in S$.
   If $x \in S$ and $y \in S$, then $xy \in S$.
   Using this definition, verify that $1101 \in S$. Solution: Basis: 1. Step 2 induction: 01. Step 3 induction: 101. Step 4 induction: 1101.

3. Let $\Sigma = \{c, d\}$, $A = \{c, dd\}$, $B = \{\Lambda, cd, d\}$, and $C = \{d\}$. Find:

   (a) AB. $\{c, dd\}$, $\{\Lambda, cd, d\}$. $AB = \{c, ccd, cd, dd, ddcd, ddd\}$.

   (b) A^2. $\{c, dd\}$, $\{c, dd\}$. $A^2 = \{cc, cdd, ddc, dddd\}$.

   (c) C^*. $C^0 \cup C^1 \cup C^2 \cup \cdots \cup C^n$ where $C^0 = \{\Lambda\}$, $C^1 = \{d\}$, $C^2 = C^1C = \{dd\}$, $C^3 = C^2C^1 = \{dd\}\{d\} = \{ddd\}$, and so on.

1.3.10 Exam 3

November 18, 1987

1. Prove by induction that $\sum_{i=1}^{n} 2^{i-1} = 2^n - 1$. Solution:
   Basis: $\sum_{i=1}^{1} 2^{i-1} = 2^1 - 1 = 2^{1-1} = 2^1 - 1 = 2^0 = 1 = 1 = 1$.
   Induction: Assume $\sum_{i=1}^{n} 2^{i-1} = 2^n - 1$ is true. Then show $\sum_{i=1}^{n+1} 2^{i-1} = 2^{(n+1)} - 1$.
   $\sum_{i=1}^{n+1} 2^{i-1} = 2^n + \sum_{i=1}^{n} 2^{i-1}$.

2. Let $\Sigma = \{c, d, g\}$ be an alphabet, and let $A = \{cg, d\}$ and $B = \{\Lambda, c, gg\}$ be languages over Σ. List the elements in the following sets.

   (a) AB. $\{cg, d\}$, $\{\Lambda, c, gg\}$. Solution: $AB = \{cg, d, cgc, dc, cggg, dgg\}$.

   (b) B^2. $\{\Lambda, c, gg\}$, $\{\Lambda, c, gg\}$.n Solution: $B^2 = \{\Lambda, c, gg, cc, cgg, ggc, gggg\}$.

   (c) A^+. $\{A' \cup A^2 \cup A^3 \cup \cdots$. $A^1 = \{cg, d\}$. Solution: $A^2 = \{cg, cg, cgd, dcg, dd\}$. $A^+ = \{cg, d, cgcg, cgd, dcg, dd, ...\}$

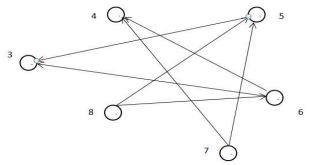

Figure 1.33: Digraph for problem 4 on Exam 3.

3. Give an inductive definition of the binary relation on $N = \{0, 1, 2, ...\}$ where $R = \{< a, b > | b = 3a\}$. $\{< 0, 0 >, < 1, 3 >, < 2, 6 >, ...\}$. Basis: $< 0, 0 > \in R$. Induction: if $< x, y > \in R$ then $< x+1, y+3 > \in R$.

4. Sketch the digraph which corresponds to $A = \{3, 4, 5, 6, 7, 8\}$ and $R = \{< x, y > | 2 \leq x - y < 4\}$. $R = \{< 8, 5 >, < 8, 6 >, < 7, 4 >, < 7, 5 >, < 6, 3 >, < 6, 4 >, 5, 3 >\}$. See Figure 1.33.

   (a) Is the digraph connected, strongly connected, or disconnected? Connected. There are arrows to each vertex in the graph. It is not disconnected because there is an undirected path to each node.

   (b) Give the incidence matrix for the digraph.

x/y	3	4	5	6	7	8
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	1	0	0	0	0	0
6	1	1	0	0	0	0
7	0	1	1	0	0	0
8	0	0	1	1	0	0

5. Let $A = \{x, y, z, w\}$. How many binary relations are there on A? Explain. Solution: $A \times A$ has $4 \times 4 = 16$ tuples. So, there are 2^{16} binary relations or subsets on A.

6. Indicate which of the properties listed are satisfied by the given digraph. For each property which is not satisfied, give an example to show it is not.

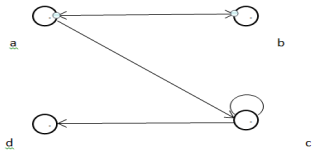

Figure 1.34: Digraph for problem 6a on Exam 3.

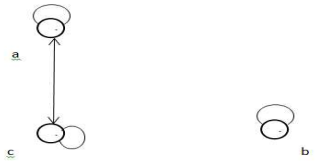

Figure 1.35: Digraph for problem 6b on Exam 3.

   (a) See Figure 1.34. reflexive. No, a is not related to itself.
   irreflexive. No, c is related to itself.

symmetric. No, $a\ R\ c$ but $c \not R\ a$.
antisymmetric. No, $a\ R\ b \wedge b\ R\ a$, but $a \neq b$.
transitive. No, $a\ R\ c \wedge c\ R\ d$, but $a \not R\ d$.

(b) See Figure 1.35. reflexive. Yes.
irreflexive. No, no node such that $x \not R\ x$.
symmetric. Yes.
antisymmetric. No, $a\ R\ c$ and $c\ R\ a$ but $c \neq a$.
transitive. Yes.

1.3.11 Quiz 5

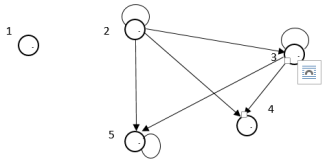

Figure 1.36: Digraph for problem 2 on Quiz 5.

1. Let $A = \{0, 1, 2, 3, 4\}$. Give the relation over A which corresponds to the predicate $P(x, y, z) \Leftrightarrow x+2y \geq z$. Solution: $R = \{< 0, 0, 0 >, < 0, 1, 2 >, < 0, 2, 4 >, < 1, 0, 1 >, < 1, 1, 3 >, < 2, 0, 2 >, < 2, 1, 4 >, < 3, 0, 3 >, < 4, 0, 4 >\}$. The L. J. Randall's comment is "there are many others."

2. Let $A = \{1, 2, 3, 4, 5\}$ and let R be the binary relation on A defined by $R = \{< x, y > | x$ is prime and $x \leq y\}$. Sketch the digraph which corresponds to $< A, R >$. Solution: See Figure 1.36.

3. Give an inductive definition of the relation of $>$ on $N = \{0, 1, 2, ...\}$. Solution: basis $< 1, 0 > \in R$. Induction: if $< x, y > \in R$, then a) $x + 1 > y$, and b) $x + 1 > y + 1$.

   (a) Use your definition to verify that $6 > 4$. Solution: basis $< 1, 0 >$. Induction
   $< 2, 1 >$
   $< 3, 2 >$
   $< 4, 3 >$
   $< 5, 4 >$
   $< 6, 4 >$

Chapter 2

Theory of Formal Languages

Instructor: Jane Randall
CS 390/590
Course Outline, Summer 1988
Office ED-249-1
Phone 440-3890, Main 3915
Office Hours: Monday, Wednesday 3:00 - 3:30pm

Course Description: This is an introduction to theoretical computer science. Topics covered will include Turing machines, foundational programming languages, computable functions, context-free grammars for formal languages, and finite automata.

Prerequisites: CS 281

Text: *Elementary Computability, Formal Languages, and Automata,* by Robert McNaughton.

Grading for 390: Your grade will be based on three tests, including the final. Each test will be 33.33% of your grade.

Make-up tests: A test may be made up only if I am contacted within 24 hours after the test is given and if the reason for absence is legitimate.

Attendance policy: Students are expected to attend class. A student who must miss class is expected to obtain the assignment and be prepared for the next class meeting.

Honor code: All students are expected to abide by the ODU Honor Code. An honor pledge will be required on all work which is to be graded.

2.1 Overview of the Course

This section gives an overview of the course. There are 3 main topics:

1. Computability — The study of what is computable and what is not; what problems can be solved using *algorithms.*
2. Formal languages — Special languages like programming languages or symbolic logic which have strict rules governing them.
3. Automata — Idealized computational models such as computers of which there is no physical instance.

The *theory of computability* is a study of problems which are solvable by algorithms and of problems which are not solvable. A *problem* is a class of questions. Each question in the class is an *instant* of the problem.

Example: Problem: What is the value of $2x + 1$ where x is some integer?

Example: Instance: What is the value of $2x + 1$ when $x = 5$?

We will assume that all questions (and answers) must be expressed in some written language like English or a programming language. We also assume that it is possible to determine whether an answer to a problem is correct or not. There are two math problem types:

1. Determine the value of a function for given values for its arguments. For example, if $f(x) = 2x + 1$, find $f(5)$.
2. Decidable problems. Problems requiring yes/no answers. For example, is $x = 3$ a solution to the equation $x^2 + 1 = 12$?

An *algorithm* for a problem is an organized set of commands for answering on demand any question that is an instance of the problem subject to the following:

1. The algorithm is written as a finite expression A, in some language.
2. Exactly which question is answered by executing the algorithm is determined by setting the *inputs* before execution begins.
3. Execution of the algorithm is a step-by-step process where the total result of the action during any one step is simple.
4. The action at each step and all results of this action are strictly determined by expression A, by the inputs, and by the results of the previous steps.
5. Upon termination, the answer to the question is a clearly specified part (the output) of the result of execution.
6. Whatever the input values, execution will terminate after a *finite* number of steps.

A *procedure* for a problem is an organized set of commands which satisfy conditions (1) thru (5) of the definition of the algorithm i.e. there may be inputs such that the procedure does not terminate after a finite number of steps. A future result will tell us that there is no method (or algorithm) for determining whether a procedure will always halt or even whether it will halt for a particular set of inputs. This is called the *halting problem.* A *non-deterministic procedure* is an organized set which satisfy conditions (1), (2), (3) and (5). A problem is *solvable* if there is an algorithm for it. Otherwise, it is *non-solvable.* A *decision procedure* is an algorithm for a class of questions for which all answers are either "yes" or "no." A problem is *decidable* if it has a decision procedure. It is *un-decidable* otherwise. Some examples of algorithms include the Euclidean algorithm for finding the greatest common divisor of two natural numbers; and the Labyrinth algorithm for determining whether a particular object is in a labyrinth.

We will represent the labyrinth by a graph. A *graph* is a ordered pair (N, E) where N is a finite set of *nodes* and E is a finite set of *edges,* each of which connects two distinct nodes. See Figure **??**. A *walk* from N_0 to

N_n is a sequence where $n \geq 0$ and E_i connects nodes N_{j-1} and N_j.

Example: $A, e_1, B, e_3, C, e_4, D$ is a walk from A to D.

A *path* from N_0 to N_n is a walk $N_0, E_1, N_1, E_2, ..., E_n, N_n$ where $N_i \neq N_j$ where $i \neq j$. All nodes are distinct in a path. A graph is *connected* if for any two nodes, there is a path between them. Some assumptions for the labyrinth problem include:

1. The graph representing the labyrinth must be connected.
2. There is an origin node labeled A.
3. One of the edges from A is named the *leading edge.*
4. If there is a target node in the labyrinth, it is labeled T.
5. Initially, no nodes or edges are colored.
6. Edges can be colored and re-colored using the colors yellow and red.

The labyrinth problem: beginning at A, travel through the graph and determine whether or not there is a node labeled T. Return to node A with the answer. See Figure **??**

2.2 Turing Machines

Alan Turing proposed Turing machines in a 1936 paper. The Turing machine, as discussed here, is a n example of a *deterministic finite state automaton.* The machine consists of a read/write head and infinite tape. The machine can assume a finite number of states and works with a finite character set. During a time cycle, the machine reads on a position on the tape and either halts or takes an action depending on what it reads and the state that it is in at the time of the read. There are at most four possible actions per time cycle:

1. Erase the symbol, just read.
2. Print a new symbol at the current position if there is no symbol already there.
3. Move one position left or right.
4. Change to a new state.

We will discuss how a Turing machine works by means of a *table* which gives all possible states, all possible characters which can be read, and the corresponding actions taken. Some notation: B represents a blank. No entry in the table means the machine halts in that situation. Execution begins at the first non-blank character. See Figure **??**.

R	Move one position to the right.
L	Move one position to the left.
$*$	Indicates start of new number.
q_i	State change.

Example: Consider the input $*//*///$. The output is $*/////$. It adds two numbers.

We will represent non-negative integers by using a unary number system.

$0 \rightarrow *$
$1 \rightarrow */$
$2 \rightarrow *//$
$3 \rightarrow *///$

When the input consists of more than one number, we write them consecutively on tape. See Figure 2.1.6 on page 28 in the textbook. Analysis:

1. The input is a string over the alphabet $\{b, c\}$.
2. An $*$ will be used to separate the input string from the copy.
3. The machine alphabet will consist of $\{b, c, *, y, z\}$.
4. As characters from the input string are copied, "b" will be changed to "y" and "c" will be changed to "z."
5. The "y's" and "z's" are changed back to "b's" and "c's" at the end.

Example: Input $\rightarrow c\ b\ b\ c$.

	c	b	b	c	*	c	b
	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$		
	q_1	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$		
$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$			
$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	q_2			
	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	
	q_3	q_c	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	
$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	q_2		
	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
	q_3		q_b			q_2	

Finish the table for homework.

Turing machines are often used to compute function values. Turing machine M computes function f with n non-negative integer arguments $i_1, i_2, ..., i_n$ if M begins with the sequence $i_1, i_2, ..., i_n$ as input and upon halting, contains *one number* on the tape which is $f(i_1, i_2, ..., i_n)$. Figure 2.1.8 in the textbook is not a function. It leaves two numbers on the output string. Figure 2.1.8 in the textbook analysis:

1. A copies the strokes of i into the positions immediately following the strokes of j.
2. As each stroke is copies, ..., etc.

Figure 2.1.9 in the textbook is multiplication. The input string $*//*///\Rightarrow *//*///*\Rightarrow */B*///*///\Rightarrow *BB*///*//////$.

Homework: On page 34 of the textbook, problems 1 thru 4, 8 thru 11. Skip Sections 2.2.1, 2.2.2, 2.2.3, and 2.2.4.

2.3 The Foundational Programming Languages

To study algorithms in terms of programs, we will develop two simplistic programming languages:

1. The Goto language.
2. The While language.

Both the Goto and While languages are *formal string languages.* The rules that govern them is very precise. To define each language, we do the following:

1. Enumerate the alphabet.
2. Define certain kinds of strings called *syntactic categories.*
3. Explain the meaning of each string in each syntactic category (semantics).

2.3.1 The Goto Language

The alphabet of the Goto language consists of 0-9, A, B, ..., Z (capitals only), :, ;, =, (,). The syntax categories are:

1. Numeral — a single digit or a string of digits which does not begin with zero.
2. Name — any string of letters and digits which begins with a letter.
3. Unlabeled command — there are 7 types.
   (a) Name := name — unlabeled variable assignment.
   (b) Name := numeral — unlabeled numerical assignment.
   (c) INCR(name) — unlabeled increment.
   (d) DECR(name) — unlabeled decrement.
   (e) GOTO name — unlabeled unconditional transfer.
   (f) If name = 0 GOTO name — unlabeled conditional transfer.
   (g) HALT — unlabeled halt.
4. Name — unlabeled command (label)
5. Program — a single command or a sequence of commands (labeled or unlabeled) seperated by semicolons. Blanks are not part of the syntax.

See Section 3.1.3 in the textbook for a discussion of parsing a program in the GOTO language. The semantics of the GOTO assumptions are:

1. The machine has unlimited storage.
2. All numbers are non-negative integers.
3. Before execution, each variable is assigned a storage location and input values are loaded into these locations.

See Section 3.2.1 in the textbook for a complete description of the meaning of each command.

Example: $DECR(x1)$ means subtract 1 from the storage location for $x1$, unless $x1$ is zero.

Notice the conditions which will cause the machine to halt. Know the three ways. A *time cycle* is the time during which the machine executes one command. A *run history* is a table showing each time cycle, the command executed, and the numbers in each storage location at the beginning and end of each time cycle. Consider Figure 3.2.1 on page 50 in the textbook which adds x, y and stores in z. Define a *proper program* in the Goto language as one satisfying:

1. For every label after a GOTO, there is exactly one labeled command in the program which has that label.
2. The last command is either a halt or an unconditional transfer. Avoid the default halt situation.

Homework, page 34, problems 1, 2,3, 4, 8, 9, 10, 11 in the textbook.

2.3.2 Homework and Answers

This is the assignment on page 34 in the textbook. Problems 1 thru 4, 8 thru 11.

These computations are similar to the copying machine of Section 2.1.3 in the textbook. In each case, design a Turing machine which, when given a string W over the alphabet $\{c, d\}$ will transform the tape into the output tape as indicated. The input string W is to be written on consecutive squares of an otherwise blank tape; and your machine is to begin in state q_1 on the leftmost nonblank square. Make sure that your machine computes correctly for every string W of length one or more over the alphabet $\{c, d\}$.

There is no restriction on how much tape you may use in the computation; or on which particular set of squares in reference to the input location the output string appears on. However, the output must appear exactly as specified, which may require that the machine "clean up" excess characters before halting. Nothing but the output should be on the output at the halt.

1. The output is the string W written backwards on consecutive squares. Solution:

State	B	c	d	z	$*$
q_1	$*q_3$	R	R		
q_2	q_4	L	L		Lq_3
q_3	Rq_5	zRq_c	zRq_d	L	L
q_c	cLq_2	R	R	R	R
q_d	dLq_2	R	R	R	R
q_4	Rq_5				
q_5				BR	BR

2. The output is a string written on consecutive squares with the same number of $c's$ and with the same number of $d's$. In other words, the output is the input string with the input characters put in alphabetical order. Solution:

1 [Place * at end
2 [loop: find c;
change to z — send to end;
restart

3 [loop: find d;
change to z — send to end;

State	B	d	c	z	$*$
a_1	$*La_2$	R	R		
a_2	Ra_4	L	zRa_c	L	L
a_c	cLa_3	R	R	R	R
a_3	a_2	L	L	L	L
a_4		R		R	La_5
a_5	a_{10}	zRa_d		L	
a_d	dLa_6	R	R	R	R
a_6		L	L		La_7
a_7	Ra_8	zRa_d		L	
a_8				BR	BR

3. The output is the string W written on the tape with a single blank space between each pair of adjacent characters. Solution:

Comment	State	B	c	d	z	/	*
Initialize	a_1	a_2	R	R			
	a_2	$*La_3$					
Reset	a_3	Ra_4	L	L	L	L	L
Put c at end	a_4		zRa_c	zRa_d	R		La_5
	a_c	ca_{c2}	R	R		R	R
	a_{c2}		R/a_3				
	a_d	da_{d2}	R	R		R	R
	a_{d2}			R/a_3			
Clean-up	a_5	Ra_6			L	L	
	a_6		R	R	BR	BR	BR

4. The output is the string W (as written as input) followed without a blank by (1)*MORE.; if there are more $c's$ than $d's$ in W. (2) *EQUAL; if there are as many $c's$ as $d's$ in W, or (3) LESS; if there are fewer $c's$ than $d's$ in W. Solution:

State	B	c	d	x	y	*	Comment
a_1	a_2	R	R				
a_2	$*La_3$						
a_3	Ra_4	L	L			L	
a_4		a_5	a_5			B	empty string
a_5		xa_6	R	R	R	a_a	more $d's$
a_6	Ra_7	L	L	L	L		
a_7		R	ya_8	R	R	a_{11}	more or equal $c's$
a_8	Ra_5	L	L	L	L		
a_9			R	R	R	Ra_{10}	less $c's$
a_{10}	Less a_{14}						
a_{11}		R	a_{12}	R	R	Ra_{13}	
a_{12}	More a_{14}	R		R	R	R	more $c's$
a_{13}	Equal a_{14}						Equal $c's$
a_{14}	Ra_{15}	L	L	L	L	L	Clean-up
a_{15}				cR	dR		

Design Turing machines to compute the following functions according to the conventions laid down in Section 2.1.4 in the textbook.

8. Absolute difference:

$$|i_1 - i_2| = \begin{cases} i_1 - i_2, & \text{If } i_1 \geq i_2. \\ i_2 - i_1, & \text{If } i_1 < i_2. \end{cases}$$

Solution:

State	B	/	*	\$	α	x	Comment
a_1	a_2	R	R				
a_2	$\$La_3$						
a_3	B	L	$\alpha La_{3,5}$				
a_4		xRa_5	R		Ra_8	R	cancel /'s
a_5		R			Ra_6		
a_6		xa_7				R	
a_7		L	a_4	La_{11}	L	L	
a_8		xRa_9				R	$i_1 > i_2$
a_9	$/a_{10}$	R		R			
a_{10}		L		L		Ra_8	
a_{11}		L	a_{12}	L	L	L	
a_{12}		xRa_{15}	R	R	Ra_{13}	R	
a_{13}	$/a_{14}$	R		R		R	
a_{14}		a_{16}					
a_{15}	$/a_{11}$	R		R	R	R	$i_1 > i_2$
a_{16}		L	a_{17}	L	L	L	clean-up
a_{17}		R	BR	*	BR	BR	

9. $QU(i_1, i_2)$ equals to the quotient when i_1 is divided by i_2. This is sometimes written as $[i_1/i_2]$ or $\lfloor i_1/i_2 \rfloor$. The quotient is the greatest integer not greater than i_1/i_2. Solution: $QU(i_1, i_2)$. $\overbrace{*///////*//*/}^{\text{bb bb}}$. $\overbrace{*/////}^{\text{bbbbb}}$ bb $\overbrace{*//}^{\text{bb}}$ $*///$.

State	B	*	α	/	x	\$	Comment
a_1	$\$La_2$	R		R			Prep string
a_2		αa_3		L			
a_3		Ra_4	L	L			
a_4		a_{11}	Ra_5	R	R		
a_5				xLa_6	R	a_8	Division
a_6			La_7		L		
a_7		a_{11}		xRa_4	L		
a_8	$/La_9$			R		R	
a_9				L		La_{10}	
a_{10}			a_4		$/L$		Reset divisor
a_{11}		BR	BR		BR	$*a_{12}$	Clean-up
a_{12}							

10. $REM(i_1, i_2)$ $*//$ $\overbrace{///}^{\text{bbb}}$ $*$ $\overbrace{///}^{\text{bbb}}$. $\overbrace{*//}^{\text{bb}}$ $bbb*$ $\overbrace{///}^{\text{bbb}}$ where b = erase.

11. $MIN(x_1, ..., x_n)$ equals to the smallest value among $x_1, x_2, ..., x_n$. Your Turing machine should work for any value of $n \geq 2$. Solution: $MIN(x1, x2)$. $*\overbrace{///}^{\text{ccc}}//*\overbrace{//}^{\text{cc}}.*\overbrace{//}^{\text{cc}}*\overbrace{//}^{\text{cc}}///*\overbrace{//}^{\text{cc}}//*\overbrace{//}^{\text{cc}}///$ where c = erase.

State	B	/	x	*	\$	y	Comment
a_1	$\$La_2$	R	R				Prep string
a_2	Ra_3	L	L				
a_3				Ra_4			
a_4		xRa_5					Find min
a_5		Ra_6		a_7	a_7		
a_6				a_3			
a_7				La_8			
a_8			yL	a_9			Put min at end
a_9				R		xa_{10}	
a_{10}	$/a_{11}$	R	R	R	R	R	
a_{11}	a_{12}	L	L	L	L	a_9	
a_{12}	Ra_{13}						Clean-up
a_{13}		BR	BR	BR	a_{14}		
a_{14}					*		

2.3.3 The While Language

The alphabet is the same as the GOTO language plus the characters >, [,]. The syntactic categories are:

1. Numeral — a single digit or a string of digits which does not begin with zero.
2. Name — any string of letters and digits which begins with a letter.
3. Command — 7 types (unlabeled).
   (a) Variable assignment — := name is an unlabeled variable assignment. Level 0 command.
   (b) Numeric assignment — Name := numeral is an unlabeled numerical assignment. Level 0 command.
   (c) Increment — INCR(name) is an unlabeled increment. Level 0 command.
   (d) Decrement — DECR(name) is an unlabeled decrement. Level 0 command.
   (e) Unilateral conditional of level $i+1$ ($i \geq 0$) : If name = 0 then [Program of level i]
   (f) Bilateral conditional of level $i+1$: If name = 0 then [Program of level i] Else [Program of level k] where $i = \max(i, k)$.
   (g) While command of level $i+1$: While name > 0 Do [Program of level i]

A program of level i is a command of level i or a sequence of commands of maximum level i separated by semicolons. Levels tell how deeply nested commands are. There are no labels in the While language. Halt is missing. Each command is executed in turn. The machine halts after the last command is executed. See Section 3.3.2 in the textbook for a description of how commands are executed. In Figure 3.3.1 on page 56 of the textbook, the second While is level 1; the first While is level 2; and Program is level 2. A While language program can easily be translated into the Goto language such that the two programs have identical flowcharts. It is not always possible to translate a Goto language program into the While language and have identical flowcharts.

Homework: page 57, problem 1 in the textbook.

2.3.4 Flowcharts

A *flowchart* is a *directed graph.* A directed graph is an ordered pair (N, A) where N is a finite set of nodes and A is a finite set of arcs or edges. Each arc goes from a node to a node (possibly the same node). A *walk* and a *path* are defined for a directed graph similar to the directions in Section 2.1. A *flowchart* is a directed graph whose nodes are circles, rectangles, and diamonds.

- Rectangle — represents a command.

- Circle — represents Begin, Halt.
- Diamond — represents a decision.

A flowchart represents a program. The language of flowcharts is a non-string formal language. Figure 3.4.1 on page 60 in the textbook is the sum of $x + y = z$. Figure 3.4.2 in the textbook is a schematic flowchart. It contains non-existent commands. Some programs may not always halt. Many functions are extremely difficult to compute. It may be hard to check that all possible situations eventually halt. Some math programs seek to find a number which satisfies a certain property without knowing whether such a number exists. Future results:

1. All computable functions can be expressed as programs in the Goto and While language.
2. There is no algorithm to tell whether a program in the Goto and While language will halt for a given input.

For homework, do problems 1, 2, 4, 5, 7, 8, 9, 10, 11, and 12 on page 64 in the textbook.

2.3.5 Homework and Answers

Page 57, problem 1 in the textbook.

1. Consider the following WHILE-language program of level 3 in which X and Y are inputs; Z is an output; and U is an auxiliary variable. Translate the program on page 58 in the textbook into a GOTO-language program, in executing which the machine will do the same things in order for each value of the inputs X and Y. Your program must have exactly the same variables, inputs, and outputs; and at the halt the value of the output must be the same. The program in this exercise computes the function $Z = f(X, Y)$ where $f(X, Y)$ is the sum of the positive terms in the sequence $2x, 2(x-3), 2(x-6), ..., 2(x-3(y-1)), (x-3y), (x-3y-1), (x-3y-2), ...$ For $Y = 0$, this sequence is $X, X-1, X-2, ...$ Solution:

```
    z:= 0
    INCR(y);
    same: If x = 0 then HALT;
    u := x;
    DECR(y);

    if y = 0 then GOTO loopif;
    DECR(x);
    DECR(x);
    DECR(x);
1   loop1: If u = 0 then GOTO same;
    DECR(u);
    INCR(z);
    INCR(z);
    GOTO loop1;

    loopif: DECR(x);
    loop2: if u = 0 then GOTO same;
    DECR(u);
    INCR(z);
    GOTO loop2;
```

2.3.6 Homework and Answers

Page 64, problems 1, 2, 4, 5, 7, 8, 9, 10, 11, and 12.

1. Give a run history in tabular form for the multiplication program in the GOTO language which was given for the flowchart of Figure 3.4.4, assuming that X, Y, W, and U are set at 3 and 2;1914, and 1939, respectfully. You will trace the multiplication of 3 by 2. Number the commands in the program 1 thru 9. Solution:

```
Time
Cycle  Line   x    y   W      U
-----------------------------------
        \     3    2  1914   1939
  1     1 ----------------------
        /     -    -   0      -
        \
  2     2 ----------------------
        /
        \     -    -   -      -
  3     3 ----------------------
        /
        \     -    1   -      -
  4     4 ----------------------
        /
        \     -    -   -      3
  5     5 ----------------------
        /
        \     -    -   -      -
  6     6 ----------------------
        /
        \     -    -   -      2
```

and so on...

In each of the following, write a program to compute the function F that is indicated. Let $x1, X2, X3, ...$ and so on, be your input variables; or simply Xif there is only one; and let Y be your output variable, with auxiliary variables as you see fit. The values of your input variables may be destroyed in the working of your program. You may give your answer as a detailed flowchart, as a GOTO-language program, or as a WHILE-language program, as you choose.

2. $F(X1, X2)$ is the maximum of two values $X1, X2$. Solution:

$$1 \left[\begin{array}{l} \text{y:= 0} \\ \text{loop: If x1 = 0 then GOTO bottom1;} \\ \text{DECR(x1);} \\ \text{if x2 = 0 then GOTO bottom2;} \\ \text{DECR(x2);} \\ \text{GOTO loop;} \\ \text{bottom1: y:= x1;} \\ \text{Halt;} \\ \text{bottom2: y:= x2;} \\ \text{Halt;} \end{array}\right.$$

5. The given function is:

$$f(x1, x2) = \begin{cases} 0, & \text{If } X1 \leq X2. \\ X1 - X2, & \text{If } X1 > X2. \end{cases}$$

Solution:

$$1 \left[\begin{array}{l} \text{if x1 = 0 then GOTO case3;} \\ \text{top: DECR(x1);} \\ \text{if x1 = 0 then GOTO case1;} \\ \text{DECR(x2);} \\ \text{if x2 = 0 then GOTO case2;} \\ \text{GOTO top;} \\ \text{case1: y:= x1;} \\ \text{DECR(x2);} \\ \text{if x2 = 0 then y:=0;} \\ \text{HALT;} \\ \text{case2: y:= x1;} \\ \text{HALT;} \\ \text{case3: y:= 0;} \\ \text{HALT;} \end{array}\right.$$

7. The given function is:

$$f(x) = \begin{cases} X, & \text{If } X \text{ is divisible by 3.} \\ X-1, & \text{If } X \text{ is not divisible by 3.} \end{cases}$$

Solution:

$$1 \left[\begin{array}{l} \text{x1:= x;} \\ \text{top: DECR(x1);} \\ \text{if x1 = 0 then GOTO bottom2;} \\ \text{DECR(x1);} \\ \text{if x1 = 0 then GOTO bottom2;} \\ \text{DECR(x1);} \\ \text{if x1 = 0 then GOTO bottom;} \\ \text{GOTO top;} \\ \text{bottom2: y:= DECR(x);} \\ \text{HALT;} \\ \text{bottom: y:=x;} \\ \text{HALT;} \end{array}\right.$$

8. The given function is:

$$f(x) = \begin{cases} 2X, & \text{If } X \text{ is even.} \\ 3X, & \text{If } X \text{ is odd.} \end{cases}$$

Solution:

```
1  x1:=x;
   top: DECR(x1);
   if x1 = 0 then GOTO x2;
   DECR(x1);
   if x1 = 0 then GOTO x3;
   GOTO top;
   x3: w:= 0; z:= 2;
   outter: if z = 0 GOTO exit1;
   DECR(z);
   u:= x;
   inner: if u = 0 GOTO outter;
   DECR(u);
   INCR(w);
   GOTO inner;
   exit1: HALT;
   x2: w:= 0; z:= 3;
   outter2: if z = 0 GOTO exit2;
   DECR(z);
   u:= x;
   inner2: if u = 0 then GOTO outter2;
   DECR(u);
   INCR(w);
   GOTO inner2;
   exit2: HALT;
```

9. The given function is:

$$f(X1, X2) = \begin{cases} 0, & \text{If } X1 = X2. \\ 1, & \text{If } X1 \neq X2. \end{cases}$$

Solution:

```
1  top: DECR(x1);
   DECR(x2);
   if x1 = 0 then GOTO bottom;
   if x2 = 0 then GOTO bottom2;
   GOTO top;
   bottom: if x2 = 0 then GOTO exityes;
   y:= 1;
   HALT;
   exityes: y:= 0;
   HALT;
   bottom2: if x1 = 0 then GOTO exit2yes;
   y:= 1;
   HALT;
   exit2yes: y:= 0;
   HALT;
```

11. $F(X1, X2)$ is the quotient when $X1$ is divided by $X2 \neq 0$. $F(X1, X2) = 0$ if $X2 = 0$. See Figure 2.1 for the solution.

12. $F(X1, X2)$ is the remainder when $X1$ is divided by $X2 \neq 0$. $F(X1, X2) = X1$ if $X2 = 0$. See Figure 2.2 for the solution.

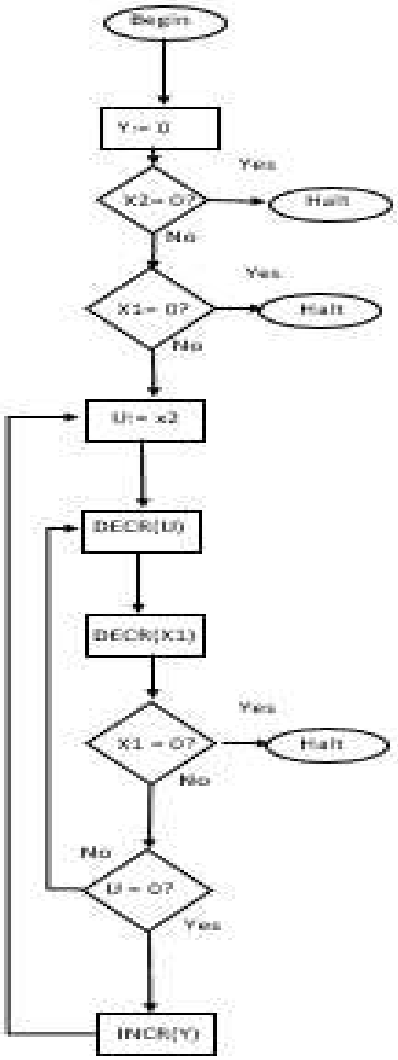

Figure 2.1: This figure shows the solution to problem 11 on page 65 in the textbook

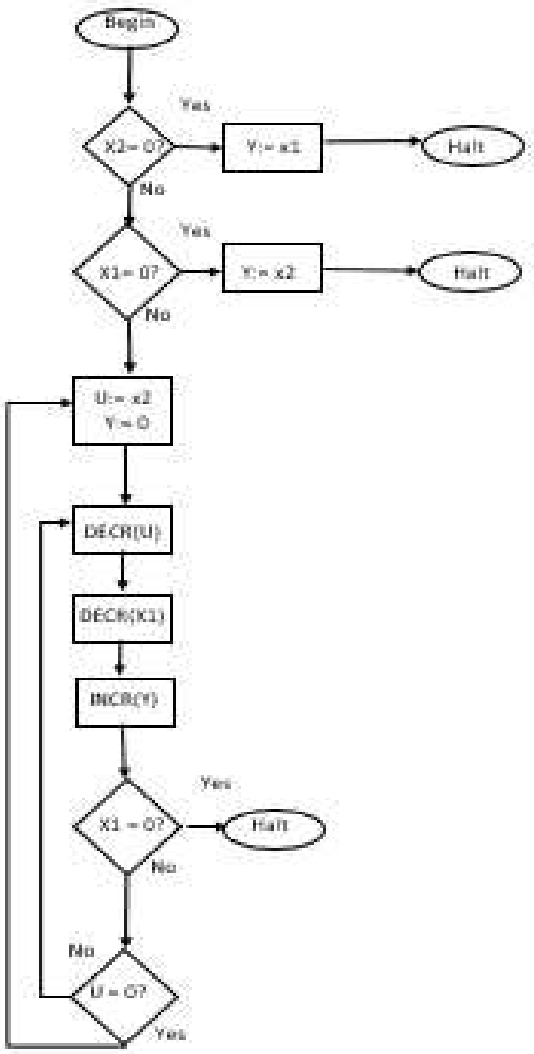

Figure 2.2: This figure shows the solution to problem 12 on page 65 in the textbook

2.3.7 Exam 1 and Answers

1. Explain the difference between an algorithm and a procedure. Solution: An algorithm always halts. A procedure may or may not halt. Procedures contain algorithms.

2. The Euclidean Algorithm produces an answer to the problem "find the greatest divisor of two positive integers x_1 and x_2."

   (a) Give an instance of this problem. Solution: $x_1 = 3$ and $x_2 = 6$. Find the GCD. The answer is 3.

   (b) Is this problem solvable or decidable? Explain. Solution: The GCD problem is solvable. It's not decidable because it does not return a yes / no answer. It is solvable because there is algorithm for it.

3. An algorithm must be deterministic. Explain what this means. Solution: Given the same set of inputs for each execution of the same algorithm, the algorithm will execute the same steps in the same order every time. To be deterministic means that the result of a step depends on expression A (the algorithm), the inputs and the results of previous steps.

4. Design a Turing machine which takes as input a non-empty string W over the alphabet $\{a, b\}$ and which produces as output string W with the first and last characters interchanged. For example, if the input is abbab, then the output is bbbaa. Give a short description of each state of your machine. Be sure to cover special cases such as strings of length one. Solution:

State	B	a	b	z	$*$	
a_1	a_2	R	R			prepare string
a_2	$*La_3$					
a_3	Ra_6	L	zRa	L	L	find all $b's$
a_4	a_5	R	R	R	R	
a_5	bL		L		a_3	
a_6		zRa_7		R	a_{10}	find all $a's$
a_7	a_8	R	R	R	R	
a_8	aLa_9					
a_9	Ra_6	L	L	L	L	
a_{10}				BL	BL	erase input

5. Write a GOTO language program to compute the function $F(X1, X2) = MIN(X1, X2)$. Use Y as your output variable.

```
1  z1:= x1;
   z2:= x2;
   top: DECR(z1);
   If z1 = 0 Then GOTO minx1;
   DECR(z2);
   If z2 = 0 then GOTO minz2;
   GOTO top;
   minx1: y:= x1;
   HALT;
   minx2: y:= x2;
   HALT;
```

6. Describe the difference between a total function and a partial function. Solution: A total function always returns one value for n-tuples; otherwise it is a partial function.

   (a) Give an example of a total function. Solution: addition.

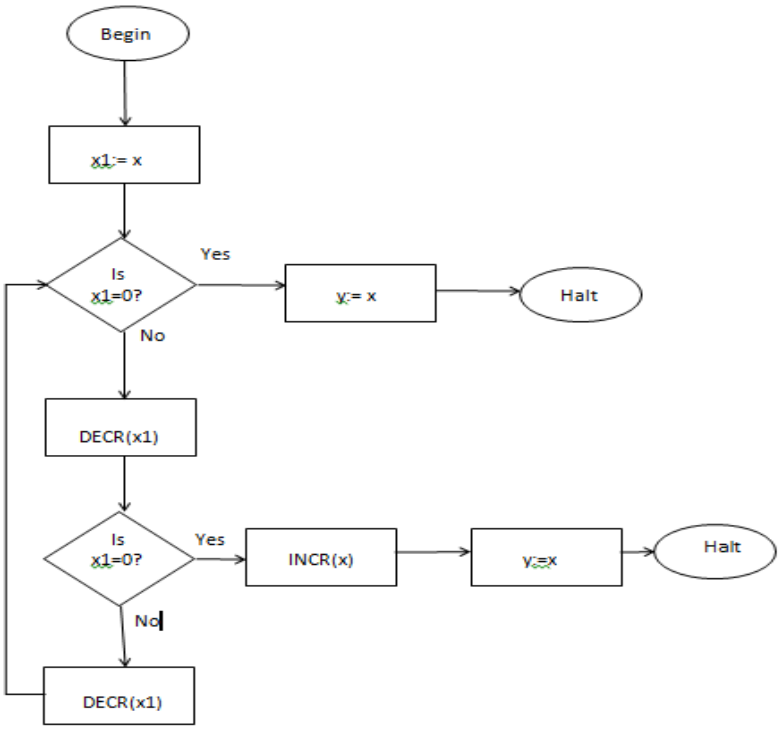

Figure 2.3: This figure shows the flowchart for problem 8 on Exam 1

(b) Give an example of a partial function. Solution: square root.

7. Write a DO-TIMES program to compute the function:

$$f(x_1, x_2) = \begin{cases} 0, & \text{If } x_1 \leq x_2. \\ x_1 - x_2, & \text{If } x_1 > x_2. \end{cases}$$

Use y as your output variable.

1
```
z1:= x1:
z2:= x2;
DO z1 TIMES
[DECR(z1);
DECR(z2);
If z2 = 0 Then [y:= z1]
Else y:= 0]
```

8. Give either a WHILE language program or a detailed flowchart to compute the function:

$$f(x) = \begin{cases} x, & \text{If } x \text{ is even.} \\ x + 1, & \text{If } x \text{ is odd.} \end{cases}$$

Use y as your output variable. Solution: See Figure 2.3.

2.4 Functions

Assume the universal set is non-negative integers. F is an n-argument function if f associates with each n-tuple $(x_1, x_2, ..., x_n)$ *at most one number*. If this number exists, it is called the *value of* f for the given

arguments. If no number exists, then the function is undefined. An argument function f is a *total function* if f always computes a value for every n-tuple. Otherwise, f is a *partial function.*

Example: The addition function is *total.*

Example: The square root function is *partial.*

A program P *computes* a function F of n arguments with respect to a sequence of $n+1$ variables if for each n-tuple $x_1, x_2, ..., x_n$ when P is run with the first n variables of S as $x_1, ..., x_n$ input. Then,

1. If $f(x_1, ..., x_n)$ has a value then the program will halt in a finite amount of time and the $(n+1)^{st}$ variable of S has value $f(x_1, ..., x_n)$ at the halt.
2. If $f(x_1, ..., x_n)$ is undefined, then program P will never halt.

Skip Section 3.6.3 Verification in the textbook. Add a new command to the While language.

- DO x TIMES [program] — execute the program the number of times equal to the value of x when the command is first executed even if x changes.

Example: $x := 4$. DO x TIMES [INCR(x)]. The program part in the DO-TIMES loop gets executed 4 times.

A DO-TIMES program is a While language which *contains no While command.* It may contain DO-TIMES commands. This results in DO-TIMES programs that always halts. Every DO-TIMES can be written as a While command.

Homework: Chapter 3, page 82 # 1, 3 in the textbook.

2.4.1 Computable Functions

We wish to be able to describe the set of computable functions over the non-negative integers. We will study four types of formal definitions for functions:

1. Explicit definition.
2. Primitive recursion.
3. Mu recursion.
4. General equational definition.

Each of these is dependent on the concept of *functional expressions.*

1. Numeral — the same as in GOTO and WHILE.
2. Variable — one of x, y or z or one of these letters followed by a numeral subscript.
3. Function symbol — one of f, g or h or one of these followed by a numeral subscript. Can also be S for successor function. Also can be any symbol introduced by a function definition.
4. Functional expression of level 0 — a numeral or a variable.
5. Functional expression of level $i+1$ — At least one funct.expr is of level i; none are of level greater than i. For example, $f(4, x1, z5, 8)$ is level 1. Another example: $f(x, y, g(x), h(y, z))$ is level 2 because x, y is level 0 and $g(x)$ and $h(y, z)$ are level 1.
6. Equation — functional expression = function expressions.

An *explicit definition* of a function with n arguments is an equation with a function symbol followed by the argument variables on the left-hand side. The right side is a functional expression with no variables except those given as arguments and using only known function symbols.

Example: Assume that multiplication and addition are known function symbols. $f(x,y) = +(*(x,y,2) \approx x*y+2$.

Example: $g(z) = *(z, S(z)) \approx z*(z+1)$.

Example: $h(x,y,z) = +(+(*(3,x),y),*(2,z)) \approx (3x+y)+2*z$.

If functions g and h are known, then a *primitive recursion* definition of the n-argument function f from g and h is a pair of equations of the form

$$\begin{cases} f(x_1, x_2, ..., x_{n-1}, 0) = g(x_1, x_2, ..., x_{n-1}) \\ f(x_1, x_2, ..., x_{n-1}, S(x_n)) = h(x_1, x_2, ..., x_n, f(x_1, x_2, ..., x_n)) \end{cases}$$

Example: To define addition,

$$\begin{cases} +(x,0) = x \\ +(x, S(y)) = S(+(x,y)) \end{cases}$$

$+(3,2) = +(3,S(1)) = S(+(3,1))$. $+(3,1) = +(3,S(0)) = S(+(3,0)) = S(3) = 4 = S(4) = 5$.

Example: To define multiplication,

$$\begin{cases} *(x,0) = 0 \\ *(x, S(y)) = +(x, *(x,y)) \end{cases}$$

Example: To define factorial,

$$\begin{cases} f(0) = 1 \\ f(S(x)) = *(S(x), f(x)) \end{cases}$$

Example: To define exponentiation,

$$\begin{cases} \uparrow(x,0) = 1 \\ \uparrow(x, S(y)) = *(x, \uparrow(x,y)) \end{cases}$$

Notice that $0^0 = 1$ by this definition.

Example: To define the predecessor function,

$$\begin{cases} P(0) = 0 \\ P(S(x)) = x \end{cases}$$

Example: To define the floored subtraction,

$$\begin{cases} \dot{-}(x,0) = x \\ \dot{-}(x, S(y)) = P(\dot{-}(x,y)) \end{cases}$$

$\dot{-}(6,2) = \dot{-}(6,S(1)) = P(\dot{-}(6,1)) = \dot{-}(6,1) = (6,S(0)) = P(\dot{-}(6,0)) = P(6) = 5 = P(5) = 4$.

Example: To define the *signum function,*

$$\begin{cases} SG(0) = & 0 \\ SG(S(x) = & 1 \end{cases}$$

where

$$\begin{cases} SG(x) = & 0 \quad \text{if } x = 0. \\ & 1 \quad \text{if } x > 0. \end{cases}$$

Example: To define the *inverse of the signum function (SG),*

$$\begin{cases} ISG(0) = & 1 \\ ISG(S(x) = & 0 \end{cases}$$

Another way to state the inverse of the signum function is

$$\begin{cases} ISG(x) = & 1 \quad \text{if } x = 0. \\ & 0 \quad \text{if } x > 0. \end{cases}$$

Some more examples using explicit definitions follows.

Example: The *absolute difference* is defined as $ABD(x, y) = +(\dot{-}(x, y, \dot{-}(y, x))$ where if $x \leq y$ then $\dot{-}(x, y) = 0$ and $\dot{-}(y, x) = y - x$. If $x > y$, then $\dot{-}(x, y) = x - y$ and $\dot{-}(y, x) = 0$.

Example: The *maximum* function is defined as $\max(x, y) = +(x, \dot{-}(y, x))$ where if $x < y$, $+(x, y - x) = x + y - x = y$ and if $x \geq y$, $+(x, \dot{-}(y, x)) = +(x, 0) = x$.

Example: The *minimum* function is defined as $\min(x, y) = \dot{-}(x, \dot{-}(x, y))$ where if $x < y$, $\dot{-}(x, 0) = x$ and if $x \geq y$, $\dot{-}(x, x - y) = x - x + y = y$.

Example: The *greater-than* function is defined as

$$GT(x, y) = SG(\dot{-}(x, y)) = \begin{cases} 1 & \text{if } x > y. \\ 0 & \text{if } x \leq y. \end{cases}$$

Example: The *greater-than or equal-to* function is defined as

$$GE(x, y) = GT(S(x), y) = \begin{cases} 1 & \text{if } x \geq y. \\ 0 & \text{if } x < y. \end{cases}$$

Example: The *equal* function is defined as

$$EQ(x, y) = *(GE(x, y), GE(y, x)) = \begin{cases} 1 & \text{if } x = y. \\ 0 & \text{Otherwise.} \end{cases}$$

We can use these basic functions to define more complex functions.

Example: *Definition by cases.*

$$f(x) = \begin{cases} 3 & \text{if } x < 3. \\ 2x & \text{if } 3 \leq x \leq 6. \\ x - 2 & \text{if } x > 6. \end{cases}$$

The above function can be defined explicitly by $f(x) = 3 * GT(3, x) + 2x * GE(x, 3) * GE(6, x) + \dot{-}(x, 2) * GT(x, 6)$.

Example: The *remainder* function. We wish to define the function $REM(x, y)$ equal to the remainder when x is divided by y. Recall if x is divided by y, then $x = y \cdot a + r$, $0 \leq r \leq y - 1$. Observation:

$$REM(x + 1, y) = \begin{cases} 0 & \text{if } REM(x, y + 1) = y. \\ REM(x, y) + 1 & \text{Otherwise.} \end{cases}$$

Due to the way primitive recursive functions are given, we cannot define the function $REM(x, y)$ this way. So, we will define a new function f by primitive recursion such that:

$$REM(x,y) = f(x,y) = \begin{cases} f(y,0) = 0 \\ f(y,S(x)) = S(f(y,x))* \\ ISG(EQ(S(f(y,x)),y)) \end{cases}$$

This remainder function $REM(S(x), y) = [REM(x, y) + 1]$.

Example: The *quotient* function is explicitly defined as $QU(x, y)$ equal to the quotient where x is divided by y. Observation:

$$QU(x+1,y) = \begin{cases} Q(x,y) & \text{if } REM(x+1,y) \neq 0. \\ Q(x,y)+1 & \text{if } REM(x+1,y) = 0. \end{cases}$$

We will define g by primitive recursion such that

$$QU(x,y) = g(y,x) = \begin{cases} g(y,0) = 0 \\ g(y,S(x)) = g(y,x) + ISG(REM(S(x),y)) \end{cases}$$

Then, $QU(x + 1, y) = QU(x, y)+$ If $REM(x + 1, y) = 0$, then $ISG(REM...) = 1$.

Do the following problems on page 98 in the textbook. # 1, 2, 3, 4, 9, 12, 17. Test on Wednesday. It covers Chapter 1 algorithm requirements (understand steps), Labyrinth and Euclidean algorithms. Chapter 3. Know commands in languages, trace programs, write programs.

2.4.2 Homework and Answers

Problems on page 98 of the textbook.

Define the function described in each exercise by using primitive recursion or explicit definition or both, from the functions listed in Section 4.2.4 in the textbook. If possible, define the function entirely by means of explicit definition.

1. $MUL(x, y) = 1$ if x is an integral multiple of y. $MUL(x, y) = 0$ if not. Note that $MUL(0, y) = 1$ but $MUL(x, 0) = 0$ if $x \neq 0$. Solution:

$$MUL(x,y) = \begin{cases} 0 & \text{if } y = 0 \text{ and } x \neq 0. \\ 1 & \text{if } x = 0 \text{ and } y \neq 0. \\ SG((REM(y,x) \end{cases}$$

2. $CON(x, y, z) = 1$ if $REM(x, z) = REM(y, z)$. $CON(x, y, z) = 0$ if $REM(x, z) \neq REM(y, z)$. Solution: $CON(x, y, z) = EQ(REM(x, z), REM(y, z))$.

3. $SD(x)$ is the sum of the positive integer divisors of x for $x \neq 0$. For example, $SD(1) = 1$. $SD(3) = 4$. $SD(6) = 12$. Solution:

$$SD(x) = \begin{cases} f(x,0) = 0 \\ f(x,y) = +(x, QU(x,S(u)) * ISG(REM(x,S(y)))) \end{cases}$$

4. $PR(x) = 1$ if x is prime. $PR(x) = 0$ otherwise. For example, $PR(0) = PR(1) = PR(4) = 0$. $PR(2) = PR(3) = PR(7) = 1$. Solution:

$$PR(x) = \begin{cases} 1 & \text{If } x \text{ is prime.} \\ 0 & \text{Otherwise.} \end{cases}$$

   x is prime iff x has exactly two divisors. $D(x)$ counts the number of divisors of x. $PR(x) = EQ(D(x), 2)$.

9. $f(0) = 1$ for all x. $f(x+1) =\uparrow (2, f(x))$ if x is even. $f(x+1) =\uparrow (f(x), 2)$ if x is odd. Solution:

$$f(x+1) = \begin{cases} \uparrow (2, f(x)), & \text{If } x \text{ is even.} \\ \uparrow (f(x), 2), & \text{If } x \text{ is odd.} \end{cases}$$

$f(0) = 1$. $f(S(x)) = ISG(REM(x, 2)) * \uparrow (2, f(x)) + SG(REM(x, 2)) * \uparrow (f(x), 2)$.

12. Solution:

$$f(x, y, z) = \begin{cases} x + y, & \text{If } 50 \leq x + y * z \leq 100. \\ z, & \text{Otherwise.} \end{cases}$$

$f(x, y, z) = (x+y) * GE(x+y*z, 50) * GE(100, x+y*z) + z * GT(50, x+y*z) + z * GT(x+y*z, 100)$.

Assume $f, f_1, f_2, ...$ are given total functions of one argument whose mathematical nature you do not know. In each case, a one argument function h is described intuitively in terms of these. Define h by means of a primitive recursion or explicit definition.

17. $h(x) = \sum_{i=0}^{x} f(x)$. Solution:

$$h(x) = \sum_{i=0}^{x} f(i) = f(0) + f(1) + \cdots + f(x).$$

Note that $h(x+1) = f(0) + h(1) + \cdots + f(x) + f(x+1)$. Then,

$$h(x) = \begin{cases} h(0) = f(0) \\ h(S(x)) = h(x) + f(S(x)) \end{cases}$$

22. $h(x) = 1$ if both $f_1(x) = 1$ and $f_2(x) = 1$. $h(x) = 0$ otherwise. Solution: $h(x) = EQ(f_1(x), 1) * EQ(f_2(x), 1)$.

2.4.3 Primitive Recursion

Recall that we defined

1. Addition

$$\begin{cases} +(x, 0) = x \\ +(x, S(y)) = S(+(x, y)) \end{cases}$$

2. Multiplication

$$\begin{cases} *(x, 0) = 0 \\ *(x, S(y)) = +(x, *(x, y)) \end{cases}$$

3. Exponentiation

$$\begin{cases} \uparrow (x, 0) = 1 \\ \uparrow (x, S(y)) = *(x, \uparrow (x, y)) \end{cases}$$

Define:

- $\Phi_0(x, y) = S(y)$.
- $\Phi_1(x, y) = +(x, y)$.
- $\Phi_2(x, y) = *(x, y)$.
- $\Phi_3(x, y) =\uparrow (x, y)$.

Similarily, define $\Phi_4(x, 0) = 1$. $\Phi_4(x, S(y)) =\uparrow (x, \Phi_4(x, y))$. Notice that $\Phi_4(2, 0) = 1$ gives the next power. $\Phi_4(2, 1) = 2$. $\Phi_4(2, 2) = 4$. $\Phi_4(2, 3) = 2^4 = 16$. $\Phi_4(2, 4) = 2^{16} = 65536$. $\Phi_4(2, 5) = 2^{65536}$. In general, $\Phi_4(x, 0) = 1$. $\Phi_4(x, 1) = x^1 = x$. $\Phi_4(x, 2) = x^x$. $\Phi(x, 3) = x^{x^x}$. and so on. Likewise, define $\Phi_5 = (x, 0) = 1$. $\Phi_5(x, S(y)) = \Phi_4(x, \Phi_5(x, y))$. Verify that $\Phi_5(2, 4) = \Phi_4(2, 65536)$. Continuing in this manner, we can define an entire sequence of functions Φ_0, Φ_1, Φ_2, ..., Φ_i, Φ_{i+1}, ... such that

$$\Phi_{i+1}(x, 0) = 1$$
$$\Phi_{i+1}(x, S(y)) = \Phi_i(x, \Phi_{i+1}(x, y))$$

Notice that for each $x \geq 2$, $\Phi_{i+1}(x, y)$ increases faster with y than $\Phi_i(x, y)$. The *class of primitive recursive functions* consists of all functions which can be divided by explicit definition or primitive recursion. $\Phi_0, \Phi_1, \Phi_2, ...$ are in the class of primitive recursive functions. All functions in the class are computable. The *Ackerman function* (A) :

$$A(0, x, y) = S(y)$$
$$A(1, x, y) = +(x, y)$$
$$A(2, x, y) = *(x, y)$$
$$A(SSS(z), x, 0) = 1$$
$$A(SSS(z), x, S(y)) = A(SS(z), x, A(SSS(z), x, y))$$

In general, $A(i, x, y) = \Phi_i(x, y)$. A is computable, but A is not in the class of primitive recursive functions (not primitive recursion and not explicit either). The trivial cases of S, $+$, $*$, and $\uparrow$ are not the same as the Ackerman function. If we define $h(x) = A(x, x, x)$ then h increases faster than any one-argument primitive-recursion function.
We wish to formally show how to compute the value of a function. We will begin with a set of equations which define the function and generate a sequence of equations which ends with the evaluation. Some definitions and notation:

- *An evaluation* — $*(3, 5) = 15$.
- *Rule of uniform substitution* — Given $*(x, S(y)) = +(x, *(x, y)) \Rightarrow *(3, S(y)) = +(3, *(3, y))$.
- *Rule of evaluation replacement* — Given $*(4, 0 = 0$, $*(4, 1) = +(4, *(4, 0)) \Rightarrow *(4, 1) = +(4, 0)$.

The fourth means to define a function: An n argument function, Q, is *equationally defined* by a set of equations, Φ, if for every n-tuple $(i_1, i_2, ..., i_n)$, no two distinct evaluations for $Q(i_1, i_2, ..., i_n)$ are derived from Φ. All explicit, primitive recursion, and mu recursion definitions can be converted to equational definitions.

2.4.4 Mu Recursion

Add μ and [] to our syntax for defining functions. A *mu expression* has the form $(\mu V)[E = 0]$ where V equals a variable and E equals an expression. The minimum value of V such that $E = 0$. A *mu recursion definition* of an f with n arguments has the form $f(x_1, x_2, ..., x_n) = (\mu V)[$ functional expression $= 0]$ where the functional expression has only $x_1, x_2, ..., x_n, V$. We will assume that the functional expression contains no mu expressions and that it contains only function symbols for total functions.

Example: $f(x, y) = (\mu z)[ISG(EQ(y * z, x)) = 0]$ i.e. $f(x, y)$ equals to the smallest value of z such that $y * z = x$. $f(8, 4) = (\mu z)[ISG(EQ(4 * z, 8)) = 0] = 2$. $f(7, 4)$ has no value.

$f(x,y) = (\mu z)[\dot{-}(x,z) = 0]$. $f(3,7) = (\mu z)[\dot{-}(3,2) = 0] = 3$, $3 - 2 = 0$.

Define $f(x,y)$ to be the least common multiple of x and y. If either $x = 0$ or $y = 0$, then let $f(x,y) = 0$. If $z = LCM(x,y)$, then z is the smallest number such that z is divisible by x and y. $f(x,y) = (\mu z)[x * y * (ISG(z) + REM((z,x) + REM(z,y)) = 0]$.

1. If x or y equal to zero, then z is zero. Consider $f(x,y) = 0$.
2. If neither x nor y equal to zero, consider $f(3,6) = (\mu z)[3 * 6 * (ISG(z) + REM(z,3) + REM(z,6)) = 0] = 6$.

f can be defined in two steps as follows:

1. Explicit — $g(x,y,z) = x * y * (ISG(z) + REM(z,x) + REM(z,y))$.
2. Mu Recursion — $f(x,y) = (\mu z)[g(x,y,z) = 0]$.

Example: Another definition for the quotient. Recall if x is divided by y, then $x = y \cdot a + r$. $\Rightarrow x \geq y \cdot a$ and $\Rightarrow y \cdot a \leq x$. So, define $QU(x,y)$ equal to the *largest* non-negative integer, w, such that $w \cdot y = x =$ the smallest z such that $z \cdot y > x - 1$. To illustrate, $QU(7,3) =$ largest w where $w \cdot 3 \leq 7 \Rightarrow w = 2$. $QU(7,3) =$ smallest z where $z \cdot > 7 - 1 \Rightarrow z = g(x,y) = (\mu z)[y * ISG(GT(z * y, x)) = 0]$. $QU(x,y) = P(g(x,y))$. $QU(x,0) = (\mu z)[0 * \cdots = 0] = 0$. $QU(x,0) = P(0) = 0$.

Since mu recursive definitions can lead to partial functions, we see that there is a relationship between mu recursion functions and the halting problem.

Example: $f(x) = (\mu z)[\uparrow (z,2) = x]$ is the partial square root function. $z = 5$ is an example.

Example: $g(x,y) = (\mu z)[ISG(z) + REM(z,x) + REM(z,y) = 0]$ is the partial positive least common multiple function. g is undefined when $x = 0$ or $y = 0$. Programs for defined functions include:

- If f is defined explicitly in terms of other functions for which the While-language programs exist, then it is possible to write a While-language program with *no new loops* which computes f. See Section 4.5.1 in the textbook.
- The previous result applies for the Goto-language, also.
- If f is defined by primitive recursion in terms of other functions for which the While-language programs exist, then it is possible to construct a DO-TIMES program which computes f. See Section 4.5.2 in the textbook.
- If f is defined by a sequence of explicit definitions and primitive recursions, then f is computable by the DO-TIMES program.
- If f is defined by mu recursion in terms of total functions which has a While-language program, then f can be computed by a While-language program. See Section 4.5.3 in the textbook. The program for f may not always halt.

Some additional results include:

- Any function which is computed by a DO-TIMES program is in the *class of primitive recursion functions.* This means it can be defined from the successor function by a sequence of explicit definitions and primitive recursions.
- Any function, partial or total, which can be computed by a While-language program can be defined by *repeated applications of explicit definitions, primitive recursions, and mu recursion.* This is the class of *mu-recursion functions.*

- A function is *mu-recursive* if it can be defined in a sequence of steps from the successor function by explicit definition, primitive recursion, and mu recursion.
- A function is *equationally definable* if it can be equationally defined in the formalism of functional expressions.

All classes of functions are equal. This is the class of computable functions.

Homework: problems on page 119, #1, 3, 5, 7.

2.4.5 Homework and Answers

Problems from page 119 in the textbook.

1. Assume as given, the function PR where $PR(x) = 1$ if x is prime; and $PR(x) = 0$ if x is not prime. (a) Define f from PR by mu recursion where $f(x)$ is the smallest prime number greater than x. (b) Define g by mu recursion, possibly with the help of explicit definition, where $g(x)$ is the greatest prime number less than x. Note that g is a partial function since $g(0), g(1)$ and $g(2)$ have no values. (c) By means of primitive recursion from f, define the one-argument function $PRIME$ where $PRIME(x) =$ the x^{th} prime in order of magnitude counting 2 as the zero-th prime. Thus $PRIME(0) = 2$. $PRIME(1) = 3$.

a. Solution: $f(x) =$ smallest prime $> x$. $f(x) = (\mu z)[ISG(PR(z) * GT(z, x)) = 0]$.

b. Solution: $g(x) =$ greatest prime $< x$. Find $z = \dot{-}(x, w)$ such that z is minimized. Note that $w = \dot{-}(x, z)$.

$$\begin{cases} h(x) = (\mu z)[ISG(PR(\dot{-}(x, z))) + ISG(z) = 0] \\ g(x) = \dot{-}(x, h(x)) \end{cases}$$

c. Solution: Prime(0) = 2. Prime(1) = 3. Prime(2) = 5. Prime(3) = 7. $f(x) =$ smallest prime $> x$. By primitive recursion, Prime(0) = 2; Prime$(S(x)) = f($Prime$(x))$.

3. Define each of the two functions by mu recursion, possibly with the help of explicit definition. (a) $g(x) = 0$ if x is a perfect square. Otherwise $g(x)$ has no value. (b) $h(x) = 0$ if x is not a perfect square. Otherwise $h(x)$ has no value.

a. Solution:

$$g(x) = \begin{cases} 0 & \text{If } x \text{ is a perfect square.} \\ \text{No value} & \text{Otherwise.} \end{cases}$$

$$\begin{cases} h(x) = (\mu z)[ISG(EQ(\uparrow (z, 2), x)) = 0] \\ g(x) = *(0, h(x)) \end{cases}$$

If x is not a perfect square, then $h(x)$ has no value.

b. Solution:

$$h(x) = \begin{cases} 0 & \text{If } x \text{ is not a perfect square.} \\ \text{No value} & \text{Otherwise.} \end{cases}$$

$z^2 < x < (z+1)^2$. $h_1(x) = (\mu z)[ISG(GT(x, z^2) * GT((z+1)^2, x)) = 0]$. $h(x) = ISG(h_1(x))$ or $z^2 > x$ and $h_1(x) = (\mu z)[ISG(GT(z^2, x)) = 0]$. L. J. Randall's comment: One idea is to look at $(z-1)^2$. Case 1: $(z-1)^2 = x$. Case 2: $(z-1)^2 \neq x$. $h(x) = (\mu z)[EQ(P(h_1(x))^2, x) = 0]$ for the case $(z-1)^2 \neq x$.

In each exercise, define the two functions described using mu recursion, possibly with the help of explicit definition, but without using primitive recursion. The function you define in part (a) of each exercise must not have a value precisely where the specification calls for it not to have a value. The function you define in part (b) must be a total function.

5a.

$$f(x,y) = \begin{cases} x+2y & \text{If } x \neq y. \\ \text{Undefined} & \text{Otherwise.} \end{cases}$$

Solution: $f(x,y) = (\mu z)[ISG(EQ(z, x+2y)) + EQ(x,y) = 0]$.

5b.

$$g(x,y) = \begin{cases} x+2y & \text{If } x \neq y. \\ 0 & \text{If } x = y. \end{cases}$$

Solution: $g_1(x,y) = (\mu z)[ISG(EQ(z, x+2y)) = 0]$. $g(x,y) = ISG(EQ(x,y)) * g_1(x,y)$.

7a.

$$f(x,y) = \begin{cases} \text{Largest } w \leq x \text{ and } h(w) = y \\ \text{No value if no } w \text{ exists.} \end{cases}$$

Solution: The smallest z will produce the largest w. $z = \dot{-}(x,w)$ or $w = \dot{-}(x,z)$. $f_1(x,y) = (\mu z)[ISG(EQ(h(\dot{-}(x,z),y))+$
$ISG(GE(x,z)) = 0]$. $f(x,y) = \dot{-}(x, f_1(x,y))$.

7b. Solution: $f_2(x,y) = (\mu z)[ISG(EQ(h(\dot{-}(x,y)),y)) * ISG(EQ(x,z)) = 0]$. $f(x,y) = \dot{-}(x, f_2(x,y))$.

2.4.6 Formal Computations Summary

A *formal computation* is a derivation of an evaluation from a given set of evaluations.

Example: The following shows a formal computation of $\dot{-}(3,1)$ for Φ.

$$\Phi = \begin{cases} \dot{-}(x,0) = x \\ \dot{-}(x,S(y)) = P(\dot{-}(x,y)) \\ P(0) = 0 \\ P(S(x)) = x \end{cases}$$

- For the class of primitive-recursive functions, these are all functions which can be defined by a sequence of explicit definitions and primitive recursions.
- Total functions — given the definition of a function in this class, we can write an algorithm to compute the function. Every function in this class is computable by a DO-TIMES program. These programs always halt.
- Not all primitive recursive functions are practically computable.
- All computable functions can be defined by a mu-recursive and an explicit definition. This is a major result.

The following four definitions are equivalent:

1. A function is *Turing computable* if it possible to construct a Turing machine that computes it.
2. A function is *program computable* if it is possible to write a program (Goto, While) that computes it.
3. A function is *mu-recursive* if it can be defined in a sequence of steps from the successor function by explicit definition, primitive recursion, and mu recursion.
4. A function is *equationally definable* if it can be equationally defined in the formalism of functional expressions.

All classes of functions are equal. This is the class of computable functions.
Omit Section 4.7 in the textbook.

Skip Chapters 5 and 6 in the textbook.

2.5 Context Free Grammars

A *grammar* is a formal description of the syntax of a formal language. A *context free grammar* consists of:

1. A set of characters called the *terminal alphabet.*
2. A set of characters called the *non-terminal alphabet,* disjoint from the set in (1).
3. The *start symbol* — a character from the non-terminal alphabet.
4. A set of *productions* of the form: non-terminal $\rightarrow$ character string, where the character string may consist of both terminal and non-terminal characters.

Example: A simplified version of the language of functional expressions.

Assume that there are:

1. Two functional symbols: S and $+$.
2. Three variables x, y, and z.
3. Binary numbers only.

The terminal alphabet is: $S, +, x, y, z, 0, 1, (,)$, comma, $<, >$. We can generate only binary numeral N with the following productions:

$N \rightarrow 0$
$N \rightarrow 1$
$N \rightarrow 1R$
$R \rightarrow 0R$
$R \rightarrow 1R$
$R \rightarrow 0$
$R \rightarrow 1$

To use a production, replace the non-terminal on the left of the arrow with the string on the right. A *derivation* to show that 1100 is a numeral is as follow:

N start symbol
$1R$
$11R$
$110R$
1100

0110 is not possible using the above productions. A derivation to show that 1100 is a numeral is as follow:

N start symbol
$1A$
$11A$
$110A$
1100

We can generate any variable V with the following productions:

$V \rightarrow x$
$V \rightarrow y$
$V \rightarrow z$

The functional expression E can be generated by:

$E \to N$
$E \to V$
$E \to +(E, E)$
$E \to S(E)$

An equation Q needs only one production: $Q \to E = E$. A derivation to show that $+(+S(x)), 11), S(0)$ is a functional expression is as follow:

E Start symbol
$+(E, E)$
$+(+(E, E), E)$
$+(+(S(E), E), E)$
$+(+(S(V), E), E)$
$+(+(S(x), E), E)$
$+(+(S(x), N), E)$
$+(+(S(x), 1A), E)$
$+(+(S(x), 11), E)$
$+(+(S(x), 11), S(E))$
$+(+(S(x), 11), ...$

This is in the textbook.

In a grammar, one type of production is chosen as being the most important. In our example, we will choose the set of Equations Q. Q will be the *start symbol* of the grammar and henceforth all derivations in the derivation will start with Q. The *language* of a context-free grammar is the set of strings over the terminal alphabet that are derivable from the start symbol. If G is a context-free grammar, then we denote the language of G by $L(G)$. The grammar defines the language. Summary:

1. Terminal alphabet $S, +, x, y, z, 0, 1, (,)$, comma.
2. Non-terminal alphabet N, A, V, E, Q.
3. Start symbol Q.
4. Productions:

$Q \to E = E$
$N \to 0|1|1A$
$A \to 0A|1A|0|1$
$V \to x|y|z$
$E \to +(E, E)|S(E)|N|V$

Page 197 in the textbook gives the full formalism of functional expressions. The more grammars, the more productions. $\{1-3\}$ are numerals. $\{4\}$ are variables. $\{5-12\}$ are functional expressions. $\{13\}$ are a sequence of functional expressions. $\{14\}$ is equations. Page 198 in the textbook gives a context free grammar for the GOTO language. $\{1-3\}$ is names. $\{4-5\}$ is unlabeled commands. $\{6\}$ is labeled commands. $\{7\}$ is programs. $\{8-10\}$ is numerals.

$\delta \to ...$
$\gamma \to ...$
$\alpha \to ...$

are the same as numerals in the previous expressions.

Example: A context-free grammar, G-IN, for propositional calculus that uses parenthesis. Assumptions:

1. Only operators are $\delta, \vee$, and $\approx$.
2. Only variables are $p, q, r, p', q', r', p'', q'', r'', ...$
3. The terminals are $\delta, \vee, \approx, p, q, r, \prime, (,)$.
4. The non-terminals are s, A.
5. The start symbol is s.
6. Productions — see page 199 in the textbook. $s \rightarrow$ and $A \rightarrow$.

Example: $p \vee \approx q$ is not possible. But $(p) \vee (\approx (q))$ is possible.

s
$(s) \vee (s)$
$(A) \vee (A)$
$(p) \vee (q)$

Example: A grammar, G-PRE, for parenthesis free propositional calculus. See page 202, bottom in the textbook. $p \vee q \approx \vee pq$.

Example: A grammar, G-Suf, for parenthesis free propositional calculus in suffix form. See page 203 in the textbook.

Example: Describe the language of the grammar. Given: $S \rightarrow cS|bD|c$. $D \rightarrow cD|bS|b$. Start symbol is S. Terminals c, b. Generate strings:

S	S
c	cS
	ccS
	$cccS$
$\vdots$	
$ccc \cdots c$	

S	S	S
bD	bD	bb
	$bccD$	$bccD$
	$bcccb$	$bcccD$
	$bcccb$	$bcccbS$
		$bcccbc$

All strings in this language have an even number of $b's$.

Example: A grammar for the set of palindromes over the alphabet $\{b, c\}$. $S \rightarrow bSb|cSc|bb|cc|b|c$. Derive $bbcbcbcbb$.

S
bSb
$bbSbb$
$bbcScbb$
$bbcbSbcbb$
$bbcbcbcbb$

Example: A grammar for $\{b^m c^n d^n e^m | m, n \geq 1\}$. $S \rightarrow bSe|bJe$. $J \rightarrow cJd|cd$. Note that $b^3 = bbb$. Derive $b^3c^2d^2e^3$.

S
bSe
$bbSee$
$bbbJeee$
$bbbcJdeee$
$bbbccddeee$

Example: A grammar for $\{b^m c^n d^n e^m f^p (gh^*)^p | m \geq 2, n \geq 0, p \geq 1\}$.

$S \rightarrow JK$
$J \rightarrow bJe|bbee|bbLee$
$L \rightarrow cLd|cd$
$K \rightarrow fKQ|fQ$
$Q \rightarrow Qh|g$

Note that h^* represents $h^n \geq 0$. Derive $b^3c^2d^2e^3f^4(gh^*)^4$.

S
JK
$bJeK$
$bbbLeeeK$
$bbbcLdeeeK$
$bbbccddeeeK$
$bbbccddeeeKfKQ$
$bbbccddeeeKffKQQ$
$bbbccddeeeKfffKQQQ$
$bbbccddeeeKffffQQQQ$
$bbbccddeeeKffffQhQQQ$
$bbbccddeeeKffffghQQQ$

2.5.1 Exam 2 and Answers

All functions on this exam are defined over the non-negative integers.

1. Define function g by explicit definition where

$$g(x,y) = \begin{cases} x+y, & \text{If } x \text{ is even.} \\ y^x, & \text{If } x \text{ is odd.} \end{cases}$$

   Solution: $ISG(REM(x,2)) * (x+y) + REM(x,2) * (\uparrow (y,x))$.

2. Define function h by primitive recursion where $h(0) = 3$ and $h(x+1) = 2 * h(x)$ if $0 < x < 10$, and $h(x+1) = 3 * h(x)$ if $x \geq 10$. Solution:

$$h(x+1) = \begin{cases} h(0) = 3 \\ h(S(x)) = GT(x,0) * GT(10,x) * 2 * h(x) + GE(x,10) * 3 * h(x) \end{cases}$$

3. Assume that f_1 and f_2 are known total functions. Define h such that:

$$h(x) = \begin{cases} 1, & \text{If } f_1(x) = 1 \text{ or } f_2(x) = 1 \text{ or both.} \\ 0, & \text{Otherwise.} \end{cases}$$

4. Give a mu-recursive definition of function f such that $f(x,y)$ is the smallest $z \leq x$ such that $h(z) = y$. $f(x,y)$ has no value if there is no such z. Assume that h is some known total function of one argument. Solution: $f(x,y) = (\mu z)[ISG(EQ(h(z),y)) + ISG(GE((x,z)) = 0]$.

5. Consider the following definition of function g. Find $g(15)$.

$$f(x) = (\mu z)[ISG(GT(x, *(2, \dot{-}(x, z)))) = 0]$$
$$g(x) = \dot{-}(x, f(x))$$

Solution:

$g(15) = \dot{-}(15, f(15) = \dot{-}(15, 8) = 7$	
$f(15) = (\mu z)[ISG(GT(15, *(2, \dot{-}(15, z)))) = 0]$	
$z = 1$	$ISG(GT(15, 2 * 14))$
	$GT(15, 28)$
$z = 2$	$ISG(GT(15, 2 * 13))$
	$GT(15, 26)$
$z = 3$	$ISG(GT(15, 2 * 12))$
	$GT(15, 24)$
$z = 4$	$ISG(GT(15, 2 * 11))$
	$GT(15, 22)$
$z = 5$	$ISG(GT(15, 2 * 10))$
	$GT(15, 20)$
$z = 6$	$ISG(GT(15, 2 * 9))$
	$GT(15, 18)$
$z = 7$	$ISG(GT(15, 2 * 8))$
	$GT(15, 16)$
$z = 8$	$ISG(GT(15, 2 * 7))$
	$GT(15, 14)$
	$ISG(1) = 0$

6. Define the class of primitive recursive functions. Given an argument to show that this class contains infinitely many functions.

7. Describe (by giving a formula) the language of the following grammar.

$$S \to bSc|H$$
$$H \to aHd|add$$

8. Give a context-free grammar for the following set of strings. $\{x^n b^{2n-1} c | n \geq 1\}$.

9. Give a context-free grammar which uses λ productions for $\{(d^*c)^n (ab^*)^{2n} | n \geq 0\}$.

2.5.2 Lambda Productions

Let λ denote the null string. If we concentrate λ with any s, we obtain s. $\lambda s = s\lambda = s$. Consider a context free grammar for $\{b^m c^n | m \geq 0, n \geq 1\}$. To generate $b^m, m \geq 0$ we can use $B \to \lambda | bB$. To generate $c^n, n \geq 1$ we use $H \to c|cH$. Now to get $b^m c^n$, we use $S \to BH$. This grammar can be written without a lambda production.

$$S \to c|bS|bH|cH$$
$$H \to c|cH$$

Example: Write a context free grammar for $\{b^m c^m h d^n e^n h f^p g^p | m, n, p \geq 0\}$.

$$S \to BhDhF$$
$$B \to bBc|\lambda$$
$$D \to dDe|\lambda$$
$$F \to fFg|\lambda$$

This grammar can be written without lambda productions.

It is always possible to convert a context free grammar with λ productions into a context free grammar with out lambda productions. If λ is in the first language, it will not be in the second language. There are three steps to convert G_0 to a grammar without lambda productions.

Step 1: Identify all non-terminals which can yield λ, either directly or indirectly. For example,

$S \to eSe$
$S \to CD$
$C \to fCg$
$C \to \lambda$
$D \to \lambda$
$B \to hB$
$B \to f$

Step 2: Construct G_1 from G_0 as follows:

1. Identify each production of G_0 which has at least one λ-yielding non-terminal on the right side.
2. Suppose the production identified in (1) has the form $B \to \cdots G_1 \cdots G_2 \cdots G_p \cdots$ where $G_1, G_2, ..., G_p$ are λ-yielding productions.
3. Add to the list of productions all those that can be formed by deleting a *non-empty subset* of $\{G_0, G_1, ..., G_p\}$ from the right side.

Step 3: Construct G_0 from G_1 by deleting. For example, for G_1,

$S \to eSe|ee$
$S \to CD|D|C|\lambda$
$C \to fCg|fg$
$C \to \lambda$

gets converted into:

$D \to BDh|Bh$
$D \to \lambda$
$B \to hB$
$B \to f$

Another example, G_2 :

$S \to eSe|ee$
$S \to CD|D|C$
$C \to fCg|fg$

gets converted into:

$D \to BDh|Bh$
$B \to hB$
$B \to f$

Do problems 1, 3, 5, 6, 7, 11 on page 212 in the text book.

2.5.3 Homework and Answers

Page 212 in the textbook.

Give a formula for the language of each of the following grammars.

1.

$S \to bbSc|H$
$H \to cHdd|cd$

Solution: $\{b^{2n}c^m d^{2m-1}e^n | n \geq 0, m \geq 1\}$.

3.

$S \to HSKK|HK$
$H \to bH|c$
$K \to Kc|d$

Solution: $\{(b^*c)^n(de^*)^{2n-1} | n \geq 1\}$.

$S \to (H)^n(K)^{2n-1}$
$H \to b^*c$
$K \to de^*$

5. $S \to bSc|bScc|bScc|bc$. Solution: $\{b^n c^m | n \geq 1, n \leq m \leq 3n-2\}$.

S
$bSccc$
$bSccccc$
$bbbccccccc$

In each exercise, give a context-free grammar for the set of strings described. Exclude the null string from consideration.

6. $\{b^{m+n}c^m d^n | m \geq 0, n \geq 1\}$. Solution:

$S \to bSd|bTd|bd$
$T \to bTc|bc$

7. $\{b^{m+n}c^m d^n | m \geq 1, n \geq 0\}$. Solution:

$S \to bSd|T$
$T \to bTc|bc$

11. $\{(d^*c)^m(c^*d)^m | m \geq 2\}$. Solution:

$S \to HSK|HHKK$
$H \to dH|c$
$K \to cK|d$

2.5.4 Homework and Answers

Page 228 in the text book.

1. Construct G_1 and G_2 where G_0 is

   $S \to bEf$
   $E \to bEc|GGc$
   $G \to b|KL$
   $K \to cKd|\lambda$
   $L \to dLe|\lambda$

   Solution: G, K, L can produce λ. G_0 is in the text book. G_1 is:

   $S \to bEf$
   $E \to bEc|GGc|Gc|c$
   $G \to b|KL|L|K|\lambda$
   $K \to cKd|cd|\lambda$
   $L \to dLe|de|\lambda$

   G_2 is

   $S \to bEf$
   $E \to bEc|GGc|Gc|c$
   $G \to b|KL|L|K$
   $K \to cKd|cd$
   $L \to dLe|de$

   Note that $\lambda \neq L(G_0)$ and that $\lambda \neq L(G_2)$.

2.5.5 Parsing

To *parse* a string means to determine the structural meaning of the string. For example, strings of a high-level programming language must be parsed to be translated into machine code. We wish to study ambiguities in formal languages. There are two types of ambiguity:

1. *Lexical ambiguity* — here a symbol or expression has more than one meaning.

2. *Structural ambiguity* — a string can be parsed into more than one way, giving it different meanings.

Formal languages should be free of structural ambiguities. If a context-free grammar is used to describe the syntax of a language, then a derivation tree can be used to parse a string of the same language. If a string has more than one derivation tree, then it has more than one structural meaning and is structurally ambiguous. A *context-free* grammar is *ambiguous* if there is a string in the language of the grammar that has at least two *non-isomorphic* derivation trees.

Example: $S \to bS|Sb|c$ is an ambiguous grammar. See Figure 2.4. Change to

$S \to bS|A$
$A \to Ab|c$

which is isomorphic.

How can we change an ambiguous grammar into an unambiguous one? One possible solution is to modify the language. This might be done by adding parenthesis or by changing to prefix notation. Suppose we do not want to modify the language to remove ambiguities. Then we must try to change the grammar for the language.

Example: $S \to bS|cS|bbS|\lambda$ is ambiguous. See Figure 2.5. The solution is $S \to bS|cS|\lambda$.

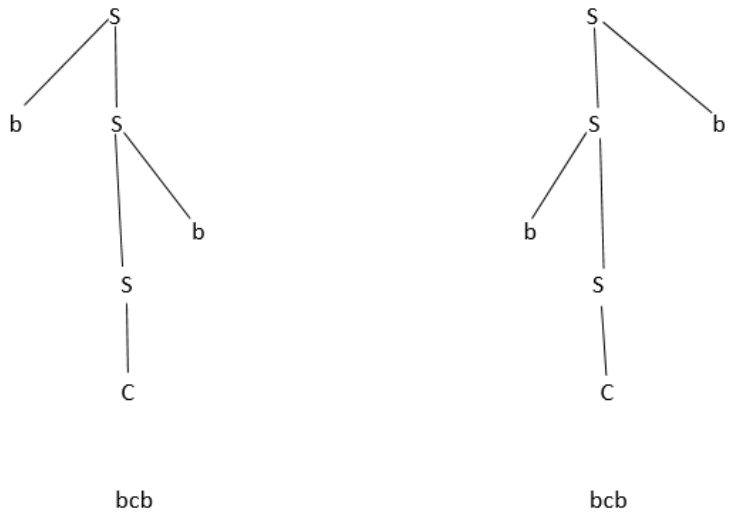

Figure 2.4: This figure shows the ambiguous grammar example

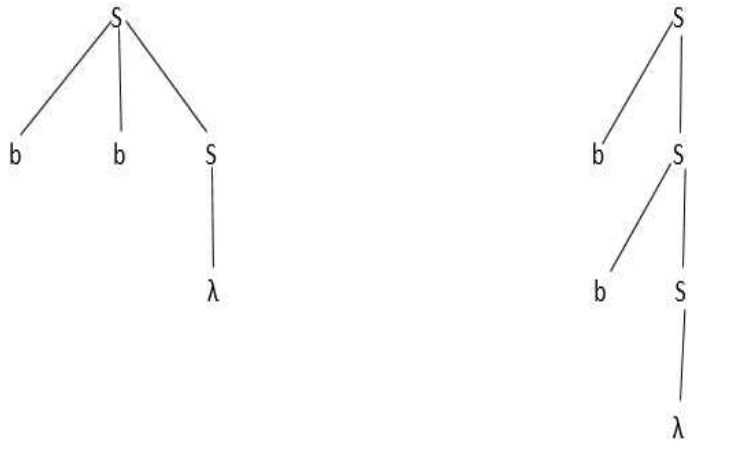

Figure 2.5: This figure shows the ambiguous grammar example

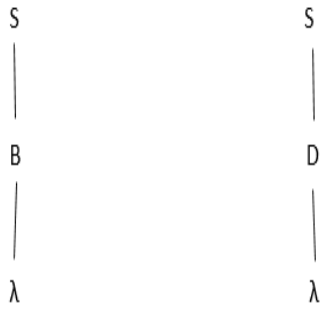

Figure 2.6: This figure shows the ambiguous grammar example

Example: The following is ambiguous

$S \to B|D$
$B \to bBc|\lambda$
$D \to dDe|\lambda$

See Figure 2.6. The solution is

$S \to B|D|\lambda$
$B \to bBc|bc$
$D \to dDe|de$

Result: Some ambiguities in a context-free grammar may not have an unambiguous equivalent. The corresponding language is said to be *inherently ambiguous.* Some more results:

1. There is no decision procedure to tell whether or not a grammar is ambiguous.
2. There is no decision procedure to tell whether or not a grammar is inherently ambiguous.
3. There is no algorithm for converting an ambiguous context free grammar into an unambiguous one.

For homework, do problems 7, 9, 11, 13, 15 on page 247 in the textbook.

2.6 Regular Languages and Finite Automata

A *regular grammar* is a context-free grammar in which every production has as its right side either

1. A terminal followed by a non-terminal.
2. A single terminal.
3. λ.

A language is regular if there is a regular grammar for it.

Example: G :

$S \to cS|bD|\lambda$
$D \to cD|bS$

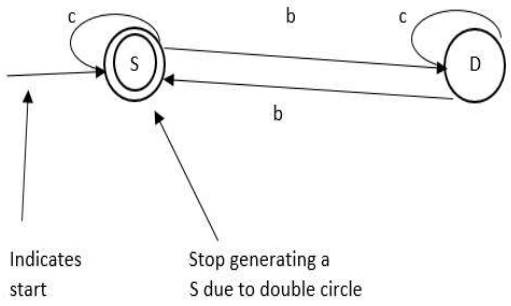

Figure 2.7: This figure shows a directed graph

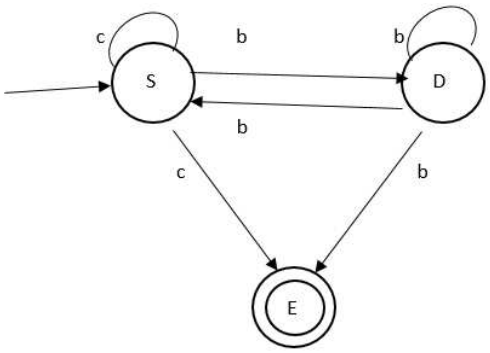

Figure 2.8: This figure shows a one-to-one correspondence

Regular grammars generate strings left to right. A *reverse regular grammar* is a context-free grammar which generates strings right to left.

Example: G' :

$S \to Sc|Db|\lambda$
$D \to Dc|Sb$

Notice that $L(G') = L(G)$.

Every regular grammar can be represented by a directed graph in which nodes correspond to non-terminals and arcs correspond to productions. See Figure 2.7.

Example: G_1 :

$S \to cS|bD|c$
$D \to cD|bS|b$

See Figure 2.8. There is a one-to-one correspondence between the walks in the graph and the strings in the grammar.

A *transition graph* is a directed graph whose arcs are labelled by characters from an alphabet. One node is the *start node.* One or more nodes are *accepting nodes* indicated by the double circles. The *language* of a

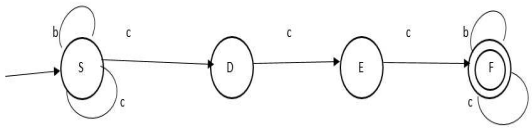

Figure 2.9: This figure shows the transition graph for the regular grammar

transition graph is the set of all strings generated by all walks from the start node to the accepting nodes. Every regular grammar has an equivalent transition graph and every transition graph has an equivalent regular grammar.

Example: Consider the regular grammar:

$S \rightarrow bS|cS|cD$
$D \rightarrow cE$
$E \rightarrow cF$
$F \rightarrow bF|cF|\lambda$

See Figure 2.9 for the transition graph.

We wish to consider a *finite automata* which is a language recognition device. A *finite automata* has a finite amount of memory. It has as input a string written on an input tape. Its goal is to determine whether the string is "acceptable." A *deterministic finite automata* consists of:

1. The machine alphabet G. These are the characters which it can read.
2. A finite set of states, K.
3. One element of K called the start state.
4. A subset of K called the accepting states.
5. A transition function $t : K \times G \rightarrow K$ where $\times$ is the Cartesian product and t maps T : (state, char) $\rightarrow$ state.

The automata begins in the start state. It reads the leftmost non-blank square on the input tape and proceeds according to the transition function. Then it goes one square to the right on the input tape. If the automata halts in an accepting state, the string is accepted; otherwise it is rejected. The language of the automata is the set of all strings over the machine alphabet that it accepts.

Example: The following defines an automata:

1. Machine alphabet $\{c, d\}$.
2. States S, Q, R.
3. Start state S.
4. Accepting state Q.
5. Transition function t :

$t(S, L) = t(Q, c) = S$
$t(S, c) = Q$
$t(Q, b) = t(R, b) = t(R, c) = R$

2.6.1 Regular Expressions

We will define a general regular expression. Each general regular expression denotes a set of strings. We will see that every general regular expression corresponds to a regular language. Let α and β be *sets* of strings. The *concatenation* of $\alpha\beta$ is given by $\alpha\beta = \{xy | x \in \alpha, y \in \beta\}$.

Example: $\alpha = \{b, bc, bcb\}$. $\beta = \{c, bb\}$. $\alpha\beta = \{bc, bbb, bcc, bcbb, bcbc, bcbbb\}$.

In general, $\alpha\beta \neq \beta\alpha$, is not commutative. If $\alpha = \{\lambda\}$, then $\alpha\beta = \beta$. If $\alpha = \emptyset$, then $\alpha\beta = \emptyset$. If α is a set of strings, then

1. $\alpha^0 = \{\lambda\}$.
2. $\alpha^{n+1} = \alpha\alpha^n$.

Example: $\alpha = \{a, b\}$. $\alpha^0 = \{\lambda\}$. $\alpha^1 = \alpha\alpha^0 = \{a, b\}$. $\alpha^2 = \alpha\alpha^1 = \{aa, ab, ba, bb\}$. $\alpha^3 = \alpha\alpha^2 = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$.

Let α be a set of strings. The *star closure* of α is denoted by α^* and is given by $\alpha^* = \bigcup_{n=0}^{\infty} \alpha^n = \alpha^0 \cup \alpha^1 \cup \alpha^2 \cdots$. $\alpha^* = G^*$ equals to all strings over G.

Example: $\alpha = \{b, bc\}$. $\alpha^* = \{\lambda, b, bc, bb, bbc, bc, bb, bbc, bcb, bcbc, ...\}$.

Definition: Let G be a string alphabet. Then,

1. $\emptyset$, $\{\lambda\}$, and $\{w\}_{w \in G}$ are general regular expressions.
2. If α and β are general regular expressions, then so are:
   (a) $(\alpha) \cup (\beta)$.
   (b) $(\alpha) \cap (\beta)$.
   (c) $(\alpha) - (\beta)$.
   (d) $(\alpha)^n$.
   (e) $(\alpha)^*$.

Notation: we will write b instead of $\{b\}$, c^* instead of $\{c\}^*$ and bc^*d instead of $\{b\}\{c\}^*\{d\}$. The precedence of operations is in this order:

1. Star, exponents.
2. Concatenation
3. Union, intersection, and difference.

Rule: parenthesis may be omitted only when operators are in different classes. For example, $\alpha^*\beta \cap \gamma$ means $(\alpha)^*(\beta) \cap \gamma$.

Example: $b^* = \{\lambda, b, bb, bbb, ...\}$ is the set of all strings over $\{b\}$.

Example: $(b \cup c)^* = \{\lambda, b, c, bb, bc, cb, cc, ...\}$

Example: $(a \cup b)^* a = \{a, b\}^*\{a\}$ is the set of all strings over $\{a, b\}$ which end in "a."

Example: $(b \cup c)^* bbb (b \cup c)^*$ is the set of all strings over $\{b, c\}$ having "bbb" as a substring.

Example: $(b \cup c)^*bbb(b \cup c)^* \cap (b \cup c)^*ccc(b \cup c)^*$ is the set of all strings over $\{b, c\}$ containing a substring "bbb" and a substring "ccc."

Example: a^*b^* is the set of all strings in which all $a's$ come before $b's$.

Example: $(aa)^* = \{\lambda, aa, aaaa, aaaaaa, ...\}$.

Example: $(a \cup c)b^* = \{a, c, abbb, ..., cbbb, ...\}$.

Example: Give a regular expression for the set of all strings of $a's$ and $b's$ of length exactly three. $(a \cup b)(a \cup b)(a \cup b) = (a \cup b)^3$.

Example: $(a \cup b)^*a(a \cup b)^*a(a \cup b)^*$.

Example: $b^*ab^*a(a \cup b)^*$.

Example: $(a \cup b)^*ab^*ab^*$.

Theorem: For a formal language L, the following are equivalent:

1. L is regular.
2. L is denoted by a restricted regular expression.
3. L is denoted by a general regular expression.

Some languages are not regular. Take the next three examples.

Example: $\{a^nb^n | n \geq 0\}$.

Example: Any language in which parentheses are used in the usual way.

Example: $\{ww^R | w \in G^*\}$ where w^R is the reverse of w.

2.6.2 Pushdown Automata

This section describes "pushdown store" or a stack. There are languages which are not context-free. For example, $\{a^nb^nc^n | n \geq 0\}$. The languages can be recognized by Turing machines.

The following are equivalent:

1. Regular grammar.
2. Reverse regular grammar.
3. Transition graph.
4. State graph.
5. Finite automata (deterministic).

Corollary: If L is a language, the following are equivalent:

1. L is regular.
2. L is the language of a transition graph.

3. L is the language of a state graph.
4. L is the language of a finite automata.

If L_1 and L_2 are regular languages, then so are $L_1 \cup L_2$ and $L_1 \cap L_2$. Note that $G^* - L$ is the set of all strings over G which are not in L. This is called the complement of L. A two-way finite automata is an automaton which can move left and right on the input tape. Result: Every two way automaton has an equivalent one-way automaton. A two-way automation is not more powerful than a one-way finite automaton.

The test will cover ambiguous grammars (2 or more); grammars: what is regular. Draw transition and state graphs. Sets. Basic results in Chapter 9 in the textbook.

A *transition graph* is a state graph iff for every node and character in the alphabet, there is exactly one arc leaving that node labeled with that character. For example, for the input table,

$$\begin{array}{ccc} b & c & \\ \uparrow & \uparrow & \uparrow \\ S & S & Q \end{array}$$

R is called the *dead* or *trapped state.*

$$\begin{array}{cccc} c & c & b & c \\ \uparrow & \uparrow & \uparrow & \uparrow \\ S & Q & Q & R \end{array}$$

The last b is rejected because it is deterministic.

State graphs correspond to finite automaton. There is exactly one arc per character. See Figure 9.2.1 in the textbook. In a *non-deterministic finite automata, t* is not a function. (Q, b) may have more than one value or no value at all. If the input string is not accepted, we can not say it should be rejected. A different execution with the input string could lead to acceptance. Every transition graph can be seen as a non-deterministic finite automata. Every non-deterministic finite automata is equivalent to deterministic finite automata. Every transition graph can be converted to a state graph. The following is an algorithm to convert a transition graph into a state graph (version 2 only).

Step 1: Begin with state Q_1, which corresponds to the start node N_1. For each character of G, find the set of nodes to which there is an arc from N_1 labeled with that character. If this set consists of N_1 only, draw an arc from Q_1 to Q_1 labeled with the character, else add a new state Q_i and draw an arc from Q_1 to Q_i.

Step i : Check if there is an arc leaving each state node labeled with each character of G. If so, we are done. Else, if Q_i is a state node with no arc labeled "w" ($w \in G$) find the set of all nodes to which there is an arc labeled "w" from a node of Q_i. If there is already a state corresponding to this set, draw an arc from Q_i to that state node and label it "w" else add a new state node and draw an arc labeled "w" from Q_i to it.

Make a state an accepting state iff it corresponds to at least one accepting node in the transition graph.

Example: See Figure 2.10 for the transition graph. See Figure 2.11 for step 1. Many times state graphs can be simplified by collapsing several states into a single state. See Figure 2.12 for the collapsed graph.

Example: The following push down automata accepts the language $\{ww^r | w \in \{a, b\}^*\}$. $G = \{a, b\}$. $R = \{a, b\}$. $K = \{S, E\}$. $F = \{E\}$.

$t(S, a, b) = (S, a) =$ push "a"
$t(S, b, \lambda) = (S, b) =$ push "b"
$t(S, \lambda, \lambda) = (E, \lambda) =$ no read
$t(E, a, a) = (E, \lambda) =$ pop "a"
$t(E, b, b) = (E, \lambda) =$ pop "b"

This machine must guess when it has reached the middle of the input string and change from state S to state E in a non-deterministic way. Whenever the machine is in state S, it can choose to either push the next symbol onto the graph or to switch to state E.

$$\begin{array}{cccc} a & b & b & a \\ \uparrow & \uparrow & \uparrow & \uparrow \\ S & S & S & S \end{array}$$

Decision: can not accept the string, but not the same as reject.

$$\begin{array}{cccc} a & b & b & a \\ \uparrow & \uparrow & \uparrow & \uparrow \\ S & S & E & E \end{array}$$

Changes states only.

For homework, on page 247 in the textbook do problems 7, 9, 11, 13, 15. On page 320, do problems 1 thru 9.

The test will cover ambiguous grammars (2 or more); grammars: what is regular. Draw transition and state graphs. Sets. Basic results in Chapter 9 in the textbook.
A *transition graph* is a state graph iff for every node and character in the alphabet, there is exactly one arc leaving that node labeled with that character. For example, for the input table,

$$\begin{array}{ccc} b & c & \\ \uparrow & \uparrow & \uparrow \\ S & S & Q \end{array}$$

R is called the *dead* or *trapped state.*

$$\begin{array}{cccc} c & c & b & c \\ \uparrow & \uparrow & \uparrow & \uparrow \\ S & Q & Q & R \end{array}$$

The last b is rejected because it is deterministic. State graphs correspond to finite automaton. There is exactly one arc per character. See Figure 9.2.1 in the textbook. In a *non-deterministic finite automata, t* is not a function. (Q, b) may have more than one value or no value at all. If the input string is not accepted, we cannot say it should be rejected. A different execution with the input string could lead to acceptance.

Theorem: The class of languages accepted by push down automata is exactly the class of context-free languages.

There are languages that are not context-free. For example, $\{a^n b^n c^n | n \geq 0\}$. These languages can be recognized by Turing machines. Final results:

- Finite automata recognize regular languages.
- Push down automata recognize context-free languages.
- Turing machines can recognize all mu-recursive functions and recognize non-context free languages.

For homework, in Chapter 8 on page 247, do problems 7, 9, 11, 13, 15. In chapter 9 on page 308, do problems 11, 12, 13, 15. In Chapter 9 on page 320, do problems 1 thru 9. The final exam will be during Final Exam week.

2.7 Homework and Answers

Page 247 in the textbook.

Show that the grammar is ambiguous with two non-isomorphic derivation trees for the same string. Find an equivalent unambiguous grammar.

7.

$S \to bSE|\lambda$
$E \to bE|\lambda$

Solution: $\{b^n|n \neq 1\}$. $S \to bE|\lambda$. $E \to bE|\lambda$.

S
bSE
$bbSEE$
$bb\lambda EEbb\lambda bE$
$bb\lambda bb$
b^4

S
bSE
$b\lambda E$
$b\lambda bE$
$b\lambda bbE$
$b\lambda bbbb$
b^4

9. $S \to bSc|bSd|bS|\lambda$. Solution: Either $S \to bSc|bSd|H|\lambda$, $H \to bH|b$ or $S \to bSc|bSd|H$, $H \to bH|\lambda$.

S
bSc
$bbSdS$
$bbb\lambda dc$
b^3dc

S
bS
$bbSc$
$bbbSdc$
$bbb\lambda dc$
b^3dc

11.

$S \to cS|cSb|Q$
$Q \to cQc|c$

Solution: $\{c^n b^m|m \geq 0, n \geq 1, n \leq m\}$. $S \to cSb|Q$. $Q \to cQ|c$.

S
cS
ccS
ccQ
ccc
c^3

S
Q
cQc
ccc
c^3

13.

$S \to bSE|\lambda$
$E \to cE|c$

Solution: $S \to bSc|bHc|\lambda$. $H \to cH|c$.

S
bSE
$bbSEE$
$bb\lambda EE$
$bb\lambda cEE$
$bb\lambda ccE$
$bb\lambda ccc$
b^2c^3

S
bSE
$bbSEE$
$bb\lambda EE$
$bb\lambda cE$
$bb\lambda ccE$
$bb\lambda ccc$
b^2c^3

15.

$S \to bS|bE$
$E \to Ed|Fd$
$F \to bFd|bddd$

Solution: $\{b^n d^m | n \geq 2, m \geq 4\}$. $S \to bS|bE$. $E \to Ed|Fd$. $F \to bddd$.

S
bE
bEd
$bEdd$
$bEddd$
$bbFdddd$
$bbbddddddd$
b^3d^7

S
bS
bbE
$bbEd$
$bbEdd$
$bbEddd$
$bbFdddd$
$bbbddddddd$
b^3d^7

The final exam is on Thursday afternoon from 12:30 to 2:30pm.

2.8 Homework and Answers

Problems on page 308 in the textbook. Do # 11, 12, 13, 15.

In Exercises 1 to 22, draw a transition graph for each set.

11. The set of strings over $\{b, c\}$ whose length is no greater than 5. Solution: See Figure 2.13.
12. The set of strings over $\{b, c\}$ whose length is at least 5. Solution: See Figure 2.14.
13. The set of all strings over $\{b, c\}$ with length at least two, having an even number (possibly zero) of $b's$. Solution: See Figure 2.15.

2.9 Homework and Answers

Problems on page 320 in the textbook.

1. Convert Figures 9.1.1, 9.1.3 to 9.1.7 and 9.1.9 to 9.1.11, respectively, into equivalent state graphs. Solution: Figure 2.16 gives the state graph for Figure 9.1.1. Figure 2.17 gives the state graph for Figure 9.1.3. Figure 2.18 gives the state graph for Figure 9.1.7 in the textbook. Figure 2.19 gives the state graph for Figure 9.1.9 in the textbook.

2.10 Handout and Answers

1. Give a regular expression for each of the following languages over the alphabet $\{a, b\}$.
   a. All strings which at some point contain a double letter (aa or bb). Solution: $G = \{a, b\}$. $(a \cup b)^*(aa \cup bb)(a \cup b)^*$.
   b. All strings which do not contain a double letter. Solution: $(a \cup b)^* - (a \cup b)^*(aa \cup bb)(a \cup b)^*$. Alternatively, $(ab)^* = \{\lambda, ab, abab, ababab, ...\}$. $(\Lambda \cup b)(ab)^*(a \cup \Lambda)$.
   c. All strings containing exactly three $b's$. Solution: $a^*ba^*ba^*ba^*$.
   d. All strings that contain the substring aaa or the substring bbb but not both. Solution: $(a \cup b)^*(aa \cup bb)$.
2. Describe the languages of the following regular expressions.
   a. $(a^*b^*)^*$.
   b. $b^*(abb^*)(\Lambda \cup a)$. Solution: Two $a's$ not adjacent. Has ab.
   c. $(ab)^*a$. Solution: It has to end in a. λ is not in the set.
   d. $a(aa)^*(\Lambda \cup a)b \cup b$. Solution: a^*b.
   e. $(a \cup b)^*a(\Lambda \cup bbbb)$. Solution: It has at least one a. Can end in a or $bbbb$. Can only be a.

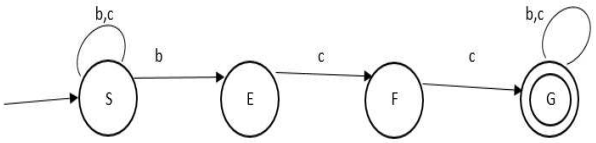

Figure 2.10: This figure shows the transition graph that needs to be simplified

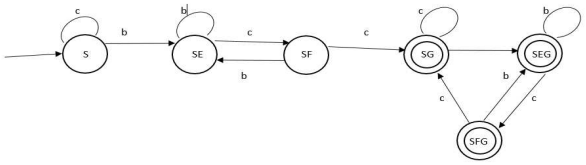

Figure 2.11: This figure shows step 1 of the simplified transition graph

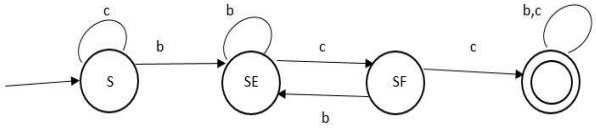

Figure 2.12: This figure shows the final step of the simplified transition graph

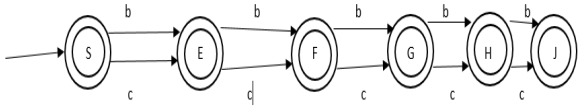

Figure 2.13: This figure shows the solution to problem 11 on page 308 in the textbook

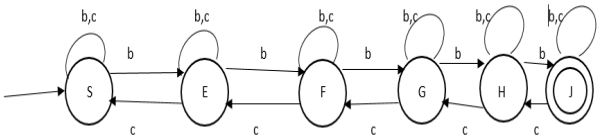

Figure 2.14: This figure shows the solution to problem 12 on page 308 in the textbook

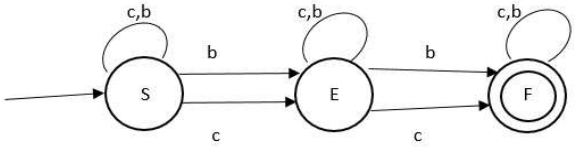

Figure 2.15: This figure shows the solution to problem 13 on page 308 in the textbook

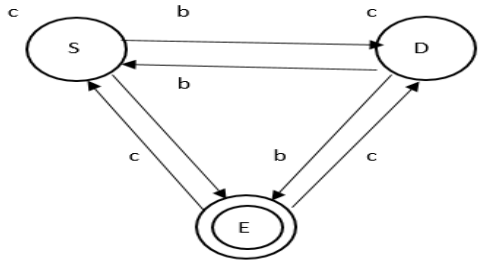

Figure 2.16: This figure shows the state graph for Figure 9.1.1 to problem 1 on page 320 in the textbook

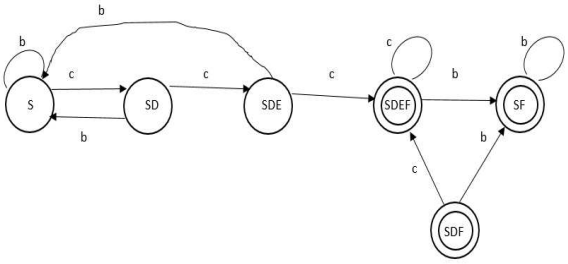

Figure 2.17: This figure shows the state graph for Figure 9.1.3 to problem 1 on page 320 in the textbook

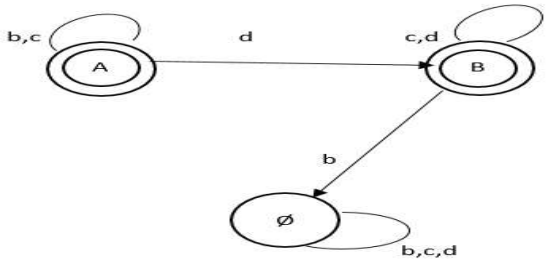

Figure 2.18: This figure shows the state graph for Figure 9.1.7 to problem 1 on page 320 in the textbook

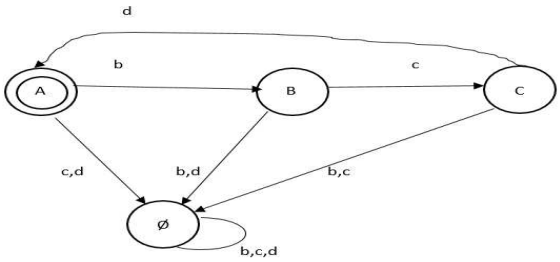

Figure 2.19: This figure shows the state graph for Figure 9.1.9 to problem 1 on page 320 in the textbook

Appendix A

Index

Index

Λ, 36
$\Re$, 41
n-tuple, 41

absolute difference function, 74
accepting node, 91
Ackerman function, 77
addition, 15, 73, 76
adjacency list, 41
Alan Turing, 57
algorithm, 56
alphabet, 36
ambiguous, 88
and, 3
antisymmetric, 44
arcs, 41
ASCII characters, 38
assertion, 7, 15, 21
assertion, sequence of, 15
associative, 26
automata, 56
axiom, 15

basis clause, 35
binary relation, 41, 42
binary tree, 37
bound variable, 8
builder notation, 23

Cartesian product, 41
change state, 57
class of primitive recursion functions, 78
command, 63
commutative, 26
complement of sets, 29
complete over, 42
complex arguments, 16
composite relation, 46
computability, 56
computable function, 64, 72, 79
compute, 72
concatenation, 36, 93
conjunction, 3, 10, 16
connected, 42, 57
constructive delimma, 16
constructive existence proof, 21
context-free grammar, 88
contigency, 5
contra-positive, 4
contradiction, 21
contridiction, 5
converse, 4
cordless path, 42
correspondence, 3

De'Morgan's laws, 32
decidable, 56
decision, 64
decision procedure, 56
definition by cases, 74
definitions for functions, 72
derivation, 81
derivation tree, 88
destructive delimma, 16
deterministic finite state automaton, 57
digraph, 41, 42
directed graph, 41, 63, 91
disconnected, 42
disjoint, 26
disjoint sets, 26
disjunction, 4
disjunctive syllogism, 15
distinct verticies, 42
distributive, 26
distributive property, 4
DO-TIMES program, 72, 78

edge, 41, 56
edges, 41
empty relation, 41
empty set, 23
empty string, 36
equal function, 74
equation, 72
equational definition, 72
equationally definable, 79, 80
equationally defined, 77
equivalence, 4
erase, 57
evaluation, 77
evaluation replacement, 77

exclusive or, 4
existential quantifier, 8
explicit, 78
explicit definition, 72, 73, 77
exponentiation, 73, 76
extremal clause, 35

factorial, 73
fallacious argument, 16
False, 3
finite automata, 92, 95, 96
finite automata, deterministic, 92
finite length string, 36
finite set, 23
first principle of mathematical induction, 37
floored subtraction, 73
flowchart, 63
formal computation, 80
formal language, 56
formal string language, 58
function symbol, 72
functional expression, 72

Goto language, 58
Goto, semantics, 59
graph, 56
graph, representation, 41
greater-than function, 74
greater-than or equal-to function, 74
greatest element, 48

halt, 64, 78
halting problem, 56, 78
hypotheses, 15
hypothetical syllogism, 16

if and only if, 7, 20
if-then, 4
iff, 7
implication, 4
incidence matrix, 41
inductive clause, 35
inductive definition, 35
inductively defined sets, 35
infinite set, 23
initial vertex, 42
INORDER, 38
input, 56
instant, question, 56
integers, set of, 8
intersection, 26, 35
inverse, 4
irreflexive, 42
is less than, 8

labyrinth, 56
labyrinth problem, 57
lambda production, 85
language, 82, 91
language over Σ, 38
largest non-negative integer, 78
leading edge, 57
least common multiple, 78
least element, 48
lexical ambiguity, 88
linear order, 47
lines, 41

math structure, 3
mathematical induction, first principle, 37
mathematical induction, second principle, 38
mathematical modeling, 3
maximum function, 74
minimum function, 74
model, reason for using, 3
modus ponens, 15
modus tollens, 15
move one position, 57
mu expression, 77
mu recursion, 72, 78
mu recursion definition, 77
mu-recursive, 79, 80
multiplication, 73, 76

natural numbers, set of, 8
negation, 3, 10
negation, properties, 10
node, 56
non empty set, 8
non-constructive existence proof, 21
non-deterministic procedure, 56
non-isomorphic, 88
non-negative integers, 57, 71, 72
non-solvable, 56
non-string formal language, 64
non-terminal alphabet, 81
null set, 23
null string, 36
numeral, 72

operation on proposition, 3
or, 4
ordered pair, 41
ordered triple, 41

pairwise disjoint sets, 26
palindrome, 83
parse, 88
partial function, 72, 78
partial order, 47

partial positive least common multiple function, 78
partial square root function, 78
partially ordered set, 47
path, 57, 63
path, length of, 42
phenomena, 3
pointer, 37
poset, 47, 48
position function, 3
power set, 35
precedence of operations, 93
predecessor function, 73
predicate, 7, 11
prediction, 3
premise, 17
primitive recursion, 72, 73
primitive recursive function, 75, 77
primitive-recursion function, 77
primitive-recursive function, 80
principle, 3
print, 57
problem, definition, 56
procedure, 56
productions, 81
program computable, 80
program, level i, 63
programming language, 56, 58
proof, 15
proof by cases, 20
proof by contradiction, 20
proof of equivalence, 20
proper program, 59
proposition, 3, 8
proposition, operations on, 3
propositional calculus, 82
push down automata, 95, 96
pushdown store, 94

quantified variable, 8
quantifier, 8, 11
quantifier, bound, 11
quantifier, existential, 8
quantifier, scope, 11
quantifier, unique existential, 8
quasi order, 47
quotient function, 75, 78

reasoning problem, 2
reflexive, 42, 46
reflexive closure, 46
regular expression, 93
regular expression, general, 94
regular expression, restricted, 94
regular grammar, 90, 94
regular grammar, reverse, 91, 94
remainder function, 74
repeated applications of explicit definitions, 78
repeated applications of mu recursion, 78
repeated applications of primitive recursions, 78
rules of inference, 15
run history, 59

scope of quantifier, 11
second principle of induction, 38
set, 23
set difference, 26
set of integers, 8
set of natural numbers, 8
set product, 38
set, elements, 23
sets, complement, 29
signum function, 73
signum function, inverse of, 74
simplification, 15
singleton, 24
smallest relation, 46
stack, 94
start node, 91
start symbol, 81, 82
state graph, 95
statement, 7
string, 36
string, empty, 36
string, finite length, 36
string, null, 36
strongly connected, 42
structural ambiguity, 88
subset, 24
subset, number of, 24
super-set, 46
symbolic logic, 56
symmetric, 44
syntactic categories, 58

table, 57
tautology, 5, 15
terminal alphabet, 81
terminal vertex, 42
terminate, 41
theory of computability, 56
there exists, 8
time cycle, 57, 59
total function, 72, 78, 80
transition graph, 91, 94–96
transitive, 44
transitive closure, 36
True, 3

truth table, 3
Turing computable, 80
Turing machine, 57, 58, 96

un-decidable, 56
unary number system, 57
unary relation, 41
undirected path, 41, 42
uniform substitution, 77
union, 26, 35
unique existential quantifier, 8
universal quantifier, 8
universal relation, 41
universe, 8

variable, 72
variable, bound, 8
variable, quantified, 8
Venn diagram, 32
vertex, initial, 42
vertex, terminal, 42
verticies, distinct, 42

walk, 56, 63
well order, 48
While language, 58, 72, 78
While language, syntactic categories, 63
Whlie language, 63
word, 36